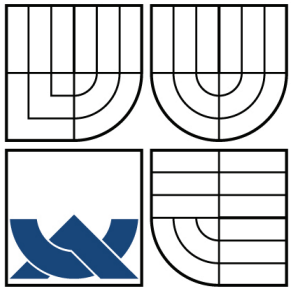
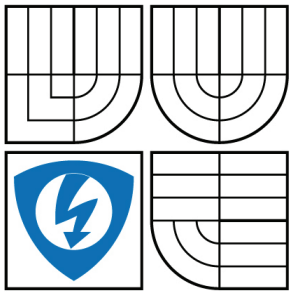


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

ROZPOZNÁVÁNÍ OBLIČEJŮ V OBRAZE

FACE RECOGNITIONS IN IMAGES

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

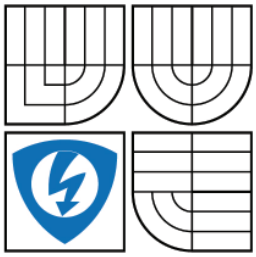
AUTOR PRÁCE — Bc. MILOŠ KRHUT

AUTHOR

VEDOUCÍ PRÁCE — Ing. KAMIL ŘÍHA, Ph.D.

SUPERVISOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Miloš Krhut	***ID:***	88586
Ročník:	2	***Akademický rok:***	2008/2009

NÁZEV TÉMATU:

Rozpoznávání obličejů v obraze

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte současné přístupy pro rozpoznání obličejů v digitálních obrazech. Vybrané algoritmy implementujte pomocí vhodného vývojového prostředí.
Doporučené vývojkové nástroje: MS Visual C++ 2008 a knihovny pro implementaci algoritmů počítačového vidění OpenCV.

DOPORUČENÁ LITERATURA:

[1] GONZALEZ R. C., WOODS R. E.: Digital Image Processing, Prentice Hall, New Jersey, 2002
[2] FISHER R. B.: CVonline: The Evolving, Distributed, Non-Proprietary, On-Line Compendium of Computer Vision, http://homepages.inf.ed.ac.uk/rbf/CVonline/
[3] LI S. Z., JAIN A. K.: Handbook of face recognition, Springer, New York, 2005

Termín zadání:	9.2.2009	***Termín odevzdání:***	26.5.2009

Vedoucí práce: Ing. Kamil Říha, Ph.D.

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Diplomová práce se zabývá tématikou detekcí obličejů v digitálních obrazech. Jsou v ní obecně popsány a roztříděny nejčastěji používané metody a zmíněny jejich výhody a nevýhody. Podrobněji je popsána metoda detekce kůže pomocí barev, detekce očí, úst a dále teoreticky popsány algoritmy strojového učení a detekce Haarovými příznaky. Dále se práce věnuje implementaci těchto metod v knihovně OpenCV, jsou zde zmíněny praktické možnosti použití a nakonec provedeno srovnání detekcí různými dostupnými natrénovanými soubory.

KLÍČOVÁ SLOVA

Detekce tváře, segmentace barev, detekce kůže, strojové učení, klasifikátory, Haarovy příznaky, OpenCV

ABSTRACT

The master thesis deals with the topic of detecting faces in digital images. There are generally described and classified the most frequently used methods and discussed their advantages and disadvantages. More detailed is described skin color detection, eye and mouth detection and are teoretically described machine learning algorithms and detection based on Haar-classifiers. The work aims to implementation of these methods in the OpenCV library, it refers to practical application of them a finally compares different provided trained files.

KEYWORDS

Face detection, color segmentation, skin detection, machine learning, classifiers, Haar-like features, OpenCV

KRHUT, M. Rozpoznávání obličejů v obraze. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 65 s. Vedoucí diplomové práce Ing. Kamil Říha, Ph.D.

Prohlášení

Prohlašuji, že diplomovou práci na téma Rozpoznávání obličejů v obraze jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

Podpis autora

Poděkování

Tímto bych rád poděkoval vedoucímu diplomové práce za užitečnou pomoc, ochotu a vstřícné jednání vedoucímu diplomové práce Ing. Kamilu Říhovi, Ph.D.

OBSAH

1 ÚVOD 9
2 POČÍTAČOVÉ VIDĚNÍ 10
2.1 Rozpoznávání vzorů 10
2.2 Rozpoznávání tváří 11
2.3 Rozdělení metod 12
2.3.1 Znalostní metody 13
2.3.2 Metody založené na invariantních rysech 13
2.3.3 Metody používající srovnávání šablon 13
2.3.4 Metody založené na zjevu 14
3 METODY INVARIANTNÍCH RYSŮ 15
3.1 Geometrické obličejové rysy 15
3.2 Barva kůže 15
3.3 Detektor kůže 16
3.3.1 Výběr barevného modelu pro detekci 16
3.3.2 Klasifikace pixelů 18
3.4 Morfologické operace 19
3.4.1 Dilatace 20
3.4.2 Eroze 20
3.5 Výběr kandidátských oblastí 23
3.6 Lokalizace obličejových částí 24
3.6.1 Lokalizace očí 24
3.6.2 Lokalizace úst 27
3.7 Využití detekce 28
4 METODY ZALOŽENÉ NA ZJEVU 29
4.1 Strojové učení 29
4.1.1 Učení bez učitele 29
4.1.2 Učení s učitelem 29
4.2 Klasifikátor 30
4.2.1 Neuronové sítě 30
4.2.2 Neuron 32
4.2.3 Support Vector Machine 33
4.2.4 Principal Component Analysis 33
4.3 Obrazové příznaky 34
4.4 Databáze tváří 34
5 DETEKCE OBLIČEJŮ POMOCÍ OPENCV 36
5.1 OpenCV 36
5.2 Detekce objektů v OpenCV 36
5.2.1 Typy Haarových příznaků 36
5.2.2 Výpočet hodnot příznaků 37
5.2.3 Kaskádové zapojení klasifikátorů 38
5.3 Učení klasifikátorů 39
5.4 Učení pomocí funkcí OpenCV 42
5.4.1 Příprava vzorků k trénování 42
5.4.2 Spuštění trénování 42

5.4.3 Soubory s natrénovanými příznaky 44
5.5 Detekce pomocí natrénovaných souborů 44
5.5.1 Načtení souboru příznaků 44
5.5.2 Detekce podle souboru 44
6 PRAKTICKÁ REALIZACE DETEKTORU 46
6.1 Detekce barvy kůže 46
6.2 Lokalizace obličejových částí pomocí barev 49
6.3 Detekce obličejů Haarovými příznaky 51
7 ZÁVĚR 59
POUŽITÁ LITERATURA 61
PŘÍLOHY 63

SEZNAM OBRÁZKŮ

Obr. 1.1: Funkce SmileShutter fotoaparátů Sony 9
Obr. 2.1: Schéma systému rozpoznávání vzorů 10
Obr. 3.1: Výsledek prahování pixelů barvy kůže 19
Obr. 3.2: Operace dilatace 20
Obr. 3.3: Binární maska obrázku 21
Obr. 3.4: Binární maska po dilataci 22
Obr. 3.5: Výsledná binární maska 22
Obr. 3.6: Příklad chybné detekce – zdrojový obrázek 22
Obr. 3.7: Binární maska chybné detekce kůže 23
Obr. 3.8: Příklad EyeMapC 26
Obr. 3.9: Příklad EyeMapL 26
Obr. 3.10: EyeMap 26
Obr. 3.11: Vyprahovaná EyeMap 26
Obr. 3.12: MouthMap 27
Obr. 4.1: Blokové schéma systému s klasifikátorem 31
Obr. 4.2: Dopředná neuronová síť 32
Obr. 4.3: Skutečný a umělý neuron 32
Obr. 4.4: Rozdělení prostoru na hyperplochy 33
Obr. 4.5: Snímání 3D povrchu tváří pro databázi MIT 34
Obr. 4.6: Ukázka tváří z Yale B databáze 35
Obr. 5.1: Typy Haarových příznaků 36
Obr. 5.2: Výpočet podokna pomocí integrálního obrazu 38
Obr. 5.3: Kaskádové zapojení klasifikátorů 39
Obr. 5.4: Slabé a silné klasifikátory 40
Obr. 6.1: Zdrojové obrázky pro detekci kůže 44
Obr. 6.2: Výběr barvy kůže prvního obrázku v RGB prostoru a v YCbCr 47
Obr. 6.3: Výběr barvy kůže druhého obrázku v RGB prostoru a v YCbCr 47
Obr. 6.4: Originál málo osvětleného obrázku 48
Obr. 6.5: Detekce málo osvětleného obrázku v RGB modelu 48
Obr. 6.6: Detekce v YCbCr modelu 48
Obr. 6.7: Portrét pro detekci očí a úst 49
Obr. 6.8: EyeMap – mapa očí 50
Obr. 6.9: MouthMap – mapa úst 50
Obr. 6.10: Detekce tváří výchozím souborem klasifikátorů 51
Obr. 6.11: Detekce výchozím souborem s upravenými parametry 52
Obr. 6.12: Detekce tváří alternativním souborem klasifikátorů 52
Obr. 6.13: Detekce tváří zepředu 55
Obr. 6.14: Detekce tváří z profilu 56
Obr. 6.15: Detekování obou typů tváří najednou 56
Obr. 6.16: Detekce tváří s nemaskovanými pixely 57
Obr. 6.17: Detekce tváří s pixely maskovanými bílou barvou 57
Obr. 6.18: Detekce očí Haarovými příznaky 58

SEZNAM POUŽITÝCH ZKRATEK

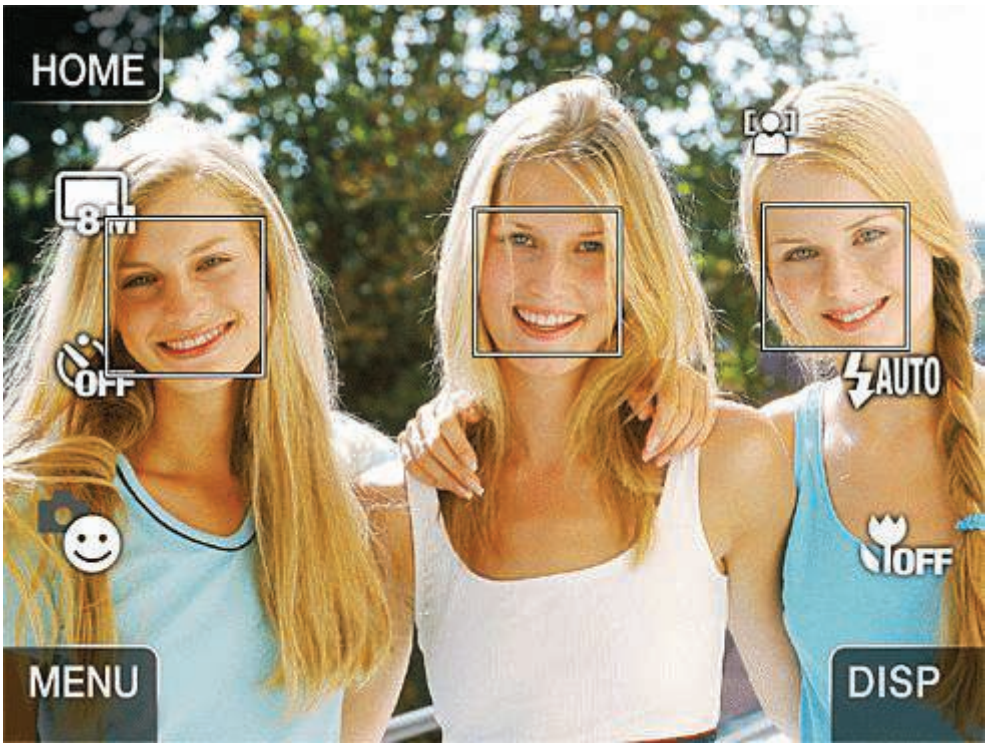
Cb	Chrominanční komponent modré barvy v modelu YCbCr
Cr	Chrominanční komponent červené barvy v modelu YCbCr
HSL	Barevný model (Hue, Saturation, Lightness)
JPEG	Joint Pictures Express Group (formát souboru obrázku)
PCA	Principal Component analysis
RGB	Red, Green, Blue (barevný model)
SVM	Support Vector Machine
Y	Jasová složka modelu YCbCr
YCbCr	Barevný model

SEZNAM DŮLEŽITÝCH SYMBOLŮ

$\cup$	Sjednocení
$\cap$	Průnik
$\oplus$	Morfologická operace dilatace
$\ominus$	Morfologická operace eroze

1 ÚVOD

Spolu s rychle se rozvíjejícími možnostmi počítačů, se zrychlujícími procesory a s obrovským pokrokem digitálního obrazové techniky dochází i k čím dál větší potřebě toho, aby stroje a počítače dokázaly „vidět“ podobně jako my, lidé. Ještě nedávno byly například automobily, které umí samy zaparkovat nebo udržovat správný směr na silnici, jen v hlavách autorů sci-fi. Dnes se však už s podobnou technologií můžeme během řízení nejnovějších luxusních aut setkat. A za pár let třeba automobil sám rozpoznápodle tváře svého majitele a odřídí cestu místo něj. Stejně tak mnoho desítek let se k pořízování fotografií používal klasický film a fotografický papír, ale trvalo jen pár let a tyhle metody byly až na pár výjimek vytlačeny digitální technikou. Pan Nicéphore Niépce, první člověk na světě, který pořídil fotografii, by se asi opravdu velmi divil při pohledu na digitální fotoaparát, který sám zaostří na tváře v záběru a třeba je i vyfotí přesně v momentě, kdy se usmějí.



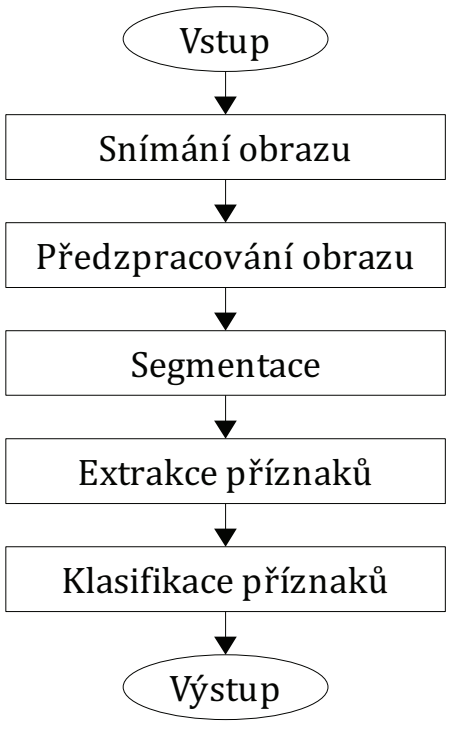
Obr. 1.1: Funkce SmileShutter fotoaparátů Sony

2 POČÍTAČOVÉ VIDĚNÍ

2.1 Rozpoznávání vzorů

Klasickým problémem počítačového vidění je nalezení specifického objektu v obraze. Ať už jde např. u automobilu o dopravní značky nebo vodící čáry u okraje silnice, nebo obyčejné rozpoznání vratné láhve v automatu v supermarketu. Nebo například už i česká dopravní policie má ve svých vozech kameru, která snímá obraz před ním, rozezná barvu auta, jeho poznávací značku a podle údajů z databáze pak počítač určí, jestli něco je nebo není v nepořádku. Tato problematika se nazývá rozpoznávání vzorů (Pattern recognition). Má za úkol buď na základě předdefinovaných nebo naučených pravidel vyhledat v obraze určitý vzor, kterým může být prakticky cokoliv, písmena textu (rozpoznávání znaků – OCR), již zmíněné SPZ vozidel, postavy lidí nebo právě jejich tváře.

Systémy pro detekci a rozpoznání vzorů by se daly obecně popsat podle následujícího obrázku:



Obr. 1.1: Schéma systému rozpoznávání vzorů

Prvním krokem je snímání obrazu, ať už kamerou, digitálním fotoaparátem, scannerem, nebo může jít i o načtení obrázku nebo videa z nějakého paměťového média (disk počítače, páska atd.). Jde v podstatě o digitalizaci, navzorkování a nakvantování intenzit složek nějakého obrazu.

Nasnímaný obraz je často třeba předzpracovat a poupravit. Nejčastěji je potřeba vyfiltrovat šum a vykompenzovat špatné světelné podmínky při snímání – patří sem třeba vyvážení bílé barvy, úprava jasu, úprava histogramu.

Poté následuje segmentace obrazu. Jde o asi nejtěžší krok, protože je při něm potřeba z obrazu vybrat a oddělit hledané objekty od zbytku. Na kvalitě segmentace pak velmi závisí kvalita dalšího zpracování. Třeba v případě dříve zmíňovaného snímacího zařízení používaného policií tento krok znamená nalezení samotného automobilu v obraze, jeho rozlišení od pozadí a okolní scény.

Dále se provede extrakce příznaků, které hledaný objekt nějakým způsobem popisují. Může jít například o SPZ automobilu, jeho barvu apod. Poté se tyto příznaky klasifikují, vyhodnocují a na jejich základě pak třídí nebo zařazují do různých kategorií.

2.2 Rozpoznávání tváří

Velmi rozsáhlou problematiku z oblasti počítačového vidění a rozpoznávání specifických vzorů tvoří rozlišování tváří. Ať už jde o samotné rozpoznání toho, že se v obraze nějaký obličej vyskytuje, nebo o rozpoznání konkrétní osoby – to se děje na základě rozlišení specifických rysů a jejich srovnání s daty v databázi. Existuje mnoho oblastí, kde jsou tyto techniky využívány. K těm nejdůležitějším patří tyto:

- **Digitální fotoaparáty a kamery** – dnešní moderní fotoaparáty (a dokonce už i některé mobilní telefony) dokáží rozpoznat tváře a správně na ně zaostřit, nastavit expozici a odstranit nebo alespoň potlačit efekt červených očí. Digitální kamery a webkamery pak dokáží tváře i sledovat a automaticky se otáčet tak, abychom byli neustále v záběru.
- **Vyhledávání osob** – policie dnes dokáže na základě fotografie nebo popisu osoby prohledat svou databázi osob a najít v ní o koho se jedná. Ale nejenom

policii mohou sloužit tyto techniky - třeba spousta lidí má v počítači uloženy tisíce fotografií, na internetu jich jsou milióny. Tak, jak je dnes běžné vyhledávat něco v obyčejném textu, tak bude v budoucnosti jednoduché na těchto fotografiích někoho vyhledat, třídit pak fotky podle toho atd. Bude stačit jen zadat jméno člověka, kterého chceme najít, nebo předložit jeho foto a vyhledávač nám ho najde. Pro zajímavost, podobnou technologii, i když zatím v menším měřítku zavádí asi nejznámější vyhledávač Google.

- **Sledovací systémy** – plánují se nasadit například na letištích. Systémy by byly napojeny třeba na databázi hledaných osob a dokázaly by v davu tyto osoby vyhledat. Podobný systém by našel využití i třeba ve vchodech do důležitých budov, bank, obchodních center, ale třeba i takový bankomat budoucnosti by mohl podle obličeje rozeznat, že z něj peníze nevybírá majitel karty, ale osoba se záznamem v trestním rejstříku. Zajímavá by byla i spolupráce těchto inteligentních sledovacích systémů s již existujícími kamerovými sítěmi ve větších městech. Stejně tak ale tyhle systému mohou sloužit k hlídání soukromého majetku, domů, parkovišť apod.

2.3 Rozdělení metod

Problematika rozpoznávání obličejů je velmi rozsáhlá. I přesto, že je poměrně mladá, existuje mnoho různých metod detekce. Přitom ne každá metoda je na určité podmínky vhodné použít. Tam, kde např. jedna metoda selhává, dává jiná velmi dobré výsledky a naopak. Podle toho, jak metody k detekci přistupují, se dají rozdělit zhruba do čtyř tříd [18]:

- Znalostní metody
- Metody založené na invariantních rysech
- Metody používající srovnávání šablon
- Metody založené na zjevu

2.3.1 Znalostní metody

U těchto metod se hledá obličej na základě předem definovaných pravidel, které ho popisují. Jedná se hlavně o základní rysy a vztahy mezi jednotlivými částmi tváře, jako např. nos, oči atd. Typickým příkladem je to, že oči jsou ve tváři symetricky, v ose mezi očima se pak nachází nos i ústa. Tyto části tváře je však třeba přesně lokalizovat, extrahovat z obrazu a přesně popsat. K tomu je potřeba často složitých a robustních algoritmů. Pokud totiž jsou definivaná pravidla příliš přísná, může algoritmus chybně zavrhnout tváře, které jim přesně neodpovídají. Pokud jsou naopak moc obecná, bude se i zvyšovat množství chybně pozitivně detekovaných tváří. Problémy těmto metodám dělají i různé pózy tváře, například při záběru z profilu. Pro tyto případy by se musely zadefinovat další pravidla, což zvyšuje již tak velkou náročnost algoritmu a může dále zvyšovat chybovost, především počet chybně pozitivních detekcí. Asi nejznámější takto založenou metodou je ta od pánů Yanga a Huanga [19].

2.3.2 Metody založené na invariantních rysech

Detekce se provádí na základě obecných rysů lidské tváře, které se s různými podmínkami (jako je světlo, natočení apod.) nemění. Hlavní myšlenkou je to, že i člověk dokáže bez problému rozeznat tváře v různých pozách a světelných podmínkách, proto musí existovat nějaké vlastnosti a příznaky, které jsou neměnné - invariantní. Tímto rysem může být třeba barva kůže nebo její textura. Pak také různé příznaky a rysy, jako třeba oči a ústa, které jsou extrahovány např. pomocí hranových detektorů. Nevýhodou je určitá náchylnost na špatné světelné podmínky (ve smyslu špatně vyvážených barev, barevném nádechu snímku apod.) nebo šum. Například i stíny způsobené špatným nasvětlením tváře mohou činit hranovým detektorům problémy. I přesto tyto metody většinou poskytují dobré výsledky a jejich implemetace není složitá. Proto se jim budu podrobněji věnovat i v kapitole 3.

2.3.3 Metody používající srovnávání šablon

Detekce využívá korelaci obrazu se šablonami buď celého obličeje nebo jeho částí. Tyto šablony jsou buď ručně předdefinovány, nebo parametrizovány vhodnou funkcí. Existence obličeje se pak rozhoduje na základě korelačních hodnot – tj. toho, jak moc obraz odpovídá vzorové šabloně. Hlavní nevýhodou je to, že je potřeba mít šablony

uložené v paměti. Taky samotné srovnávání může být časově náročné, stejně tak vytváření a připravování šablon.

2.3.4 Metody založené na zjevu

Tyto metody, narozdíl od předchozích, nevyužívají předem nadefinované šablony pro základní vlastnosti a rysy obličeje, ale jsou odvozeny automatickým strojovým učením z ukázkových obrazů. Strojové učení je uzpůsobeno tak, aby si pamatovalo typické rysy a charakteristiky tváře a zároveň dokázalo rozlišit i obrazy, které obličejem nejsou. Metod tohoto typu je velké množství, dosahují často velmi dobrých výsledků, například SVN [14], metoda „vlastních tváří“ (Eigenfaces) [21] nebo Hidden Markov Model.

Vzhledem k podávaným výsledkům jednotlivých metod a jejich náročnosti na implementaci jsem se rozhodl dále se věnovat jen metodám založeným na invariantních rysech a metodám založeným na zjevu. Hlavně ty poslední zmíněné jsou nejperspektivnější a asi nejrychleji se rozvíjející.

3 METODY INVARIANTNÍCH RYSŮ

Invariantní rysy jsou takové rysy, které jsou neměnné a nezávislé na parametrech scény nebo na různorodosti tváří jednotlivých lidí. Těchto rysů je několik druhů. Některé metody zkoumají geometrické uspořádání části obličeje, nejčastěji očí, nosu a úst. Jiné jsou založeny na odlišnosti textury lidské kůže nebo její barvy od okolí.

3.1 Geometrické obličejové rysy

Tyto metody hledají přímo základní obličejové rysy tváře, oči, obočí, nos a ústa. Pro jejich nalezení a zvýraznění vůči zbytku obrazu se využívají hranové detektory, popřípadě různé filtrační metody. Dále mohou vycházet ze znalostí, kde se jednotlivé rysy nacházejí. To se dá použít jak pro zrychlení detekce (např. po detekci očí se dá předpokládat, že ústa budou někde pod nimi, v určité vzdálenosti vzhledem ke vzdálenosti očí), tak pro samotné ověření úspěšné detekce (např. pokud se nos nenachází v linii mezi očima, jde pravděpodobně o chybnou detekci a je třeba ji zamítnout).

V případě hodně komplexního prostředí, ve kterém se obličej nachází, tato metoda často selhává. Hranový detektor detekuje příliš mnoho hran, může zde být příliš mnoho objektů, které obličejové rysy zdánlivě připomínají a způsobují chybné pozitivní detekce.

3.2 Barva kůže

Hledání barvy kůže je velmi rychlá a přitom jednoduchá metoda. I proto je velmi často využívána, i když často jen jako pomocná metoda nebo doplněk složitějších detektorů. Metoda je ve velké míře odolná vůči změnám světelných podmínek, hlavně vůči intenzitě jasu, méně už pak vůči špatnému vyvážení barev. Naprosto nezávislá je pak na velikosti tváře, jejímu natočení, na věku a pohlaví dané osoby a také na barvě pleti. Umožňuje také často velmi dobře detekovat tváře s různými strukturálními komponenty obličeje, jako vousy, brýle, vlasy zakrývající část tváře apod.

Nevýhodou je jednak to, že detekuje opravdu všechnu kůži, ne jen tu na obličeji. Často tak za obličej může označit třeba holé ruce, krk, popřípadě další různé odhalené části těla, které jsou na obrázku. Další nevýhodou je to, že pokud se v obraze v pozadí nebo okolí nachází místo, které má barvu velmi podobnou barvě kůže, je i toto místo za kůži označeno. Proto je třeba tyto nežádoucí detekce nějak potlačit, nejčastěji pomocí morfologických operací, které popíšu později.

3.3 Detektor kůže

Princip metody vychází z toho, že barva lidské kůže zabírá v barevných modelech vždy určitý daný prostor. Ohraničením tohoto prostoru lze pak jednoduše klasifikovat barvy jednotlivých pixelů obrazu, tj. jestli se v prostoru nachází a lze je považovat za kůži nebo ne [12].

3.3.1 Výběr barevného modelu pro detekci

RGB

V počítačové grafice se využívá mnoho různých barevných modelů. Základním modelem nejčastěji používaným při zpracování obrazu je RGB. Zde je každá barva definována intenzitami třech různých složek – R (Red, červená), G (Green, zelená) a B (Blue, modrá). U tohoto modelu tedy není žádné rozlišení na jasovou a barevnou složku, proto je většinou pro detekci kůže hůře použitelný.

Normalizovaný RGB

Tento barevný prostor se od klasického RGB odlišuje zásadní vlastností – a to že součet všech tří složek je roven jedné (r + g + b = 1). Díky tomu lze jednu ze složek vypustit (lze ji snadno vypočítat jako zbytek do 1) a redukovat tím barevný prostor. Zbylé složky se pak nazývají čisté barvy, protože normalizací byla snížena jejich závislost na jasu, což je hlavně v případě detekcí tváří výhoda (jas zde ale ovšem neodpovídá plně tomu, který vnímá oko). Složky prostoru vypočítáme z klasických RGB dle rovnice 3.1.

$$r = \frac{R}{R+G+B} \quad g = \frac{G}{R+G+B} \quad b = \frac{B}{R+G+B} \tag{3.1}$$

YCbCr

V televizní technice a např. i ve formátu JPEG k ukládání obrázků se používá barevný model YCbCr [18]. Y zde představuje jasovou složku, Cb a Cr jsou tzv. chrominanční složky (Cb – chrominanční komponent modré a Cr – chrominanční komponent červené barvy). Model YCbCr lze ze základního RGB získat např. pomocí tohoto vztahu:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 65{,}481 & 128{,}553 & 24{,}966 \\ -37{,}797 & -74{,}203 & 112{,}000 \\ 112{,}000 & -93{,}786 & -18{,}214 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{3.2}$$

U modelu YCbCr je tedy jasová složka oddělená od barevných, tím pádem při snímání obrazu nezáleží na jeho jasu, ale jen na barevném podání – a právě barevné složky jsou ty, pomocí kterých můžeme oddělit barvy kůže od ostatních.

HSL

Posledním z nejčastěji používaných barevných modelů je HSL. Má opět tři složky: Hue (odstín), Saturation (sytost barev) a Lightness (jas, světelnost). Opět je zde tedy oddělena jasová složka od ostatních, navíc tento model je podobný lidskému vnímání barev.

Způsobů, jak vypočítat jednotlivé složky HSL je více. Asi nejjednodušší z nich je tento:

$$H = arcos \frac{\frac{1}{2}((R-G)+(R-B))}{\sqrt{(R-G)^2+(R-B)(G-B)}} \tag{3.3}$$

$$S = 1 - 3\frac{min(R, G, B)}{R+G+B} \tag{3.4}$$

$$V = \frac{1}{3}(R+G+B) \tag{3.5}$$

Ideální jsou tedy modely s oddělenou jasovou složkou – YCbCr a HSL. Na základě pokusů a taky díky daleko jednodušší konverzi z RGB jsem se rozhodl pro YCbCr.

3.3.2 Klasifikace pixelů

Po transformaci do YCbCr barevného prostoru je třeba vybrat vhodný způsob klasifikace toho, zda daná barva pixelu náleží mezi barvy kůže či nikoliv. Tyto klasifikátory mohou být buď neparametrické nebo parametrické a mohou být definovány už předem, nebo výběrem z nějakých experimentálních vzorových šablon.

U neparametrických modelů kůže není její barva zadána explicitně pomocí určitých pravidel, ale existuje pravděpodobnostní nebo binární mapa pro každou hodnotu barvy. Pravděpodobnostní mapa pak určuje míru pravděpodobnosti, že jde o barvu kůže [7], binární pouze informaci, zda o kůži jde nebo nikoliv. Tyto mapy je vetšinou předem připraveny z trénovací vzorové množiny. Výhodou tohoto přístupu je o něco vyšší rychlost detekce, nevýhodou zase někdy paměťové nároky. Také může být pracné tyto šablony připravit.

Parametrické modely kůže naproti tomu popisují rozložení barvy kůže určitým matematickým modelem. Jeho parametry jsou získány buď opět z nějaké trénovací množiny, nebo jsou zadány přímo, na základě experimentálních výsledků. Dá se třeba využít poznatku, že ve většině barevných prostorů má podprostor barvy kůže eliptický tvar.

Velmi dobrých výsledků lze dosáhnout i jednoduchým rozlišením, jestli jednotlivé složky barvy patří do určitého intervalu či nikoliv, v podstatě tedy prahováním hodnot. Testováním a srovnáváním cizích výsledků jsem došel k těmto intervalům a podmínkám, které vykazovaly nejlepší výsledky [12, 20].

- **Pro RGB model:**

$$\begin{aligned} &R>95, G>40, B>20 \\ &max(R,G,B)-min(R,G,B)>15 \\ &|R-G|>15 \\ &R>G, R>B \end{aligned} \tag{3.6}$$

- **Pro model YCbCr:**

$$\begin{aligned} &Y \in <77,127> \\ &Cb \in <133,173> \\ &Cr \in <20,230> \end{aligned} \tag{3.7}$$

Příklad výsledku prahování a nastavení všech barev neodpovídajících kůži na černou můžeme vidět na obrázku 3.1.

Obr. 3.1: Výsledek prahování pixelů barvy kůže

3.4 Morfologické operace

Výstupem klasifikátoru, který rozhoduje, zda dané pixely obrázku patří nebo nepatří do množiny barev kůže, je vetšinou bitová maska. V ní se ovšem s největší pravděpodobností vyskytují chyby – buď jsou nějaké barvy chybně označeny za kůži, většinou v pozadí o podobné barvě, nebo naopak jsou oblasti v tváři chybně označeny za nevyhovující – to se stává třeba u očí; ty mají jinou barvu než kůže a přitom do obličeje patří také. Tyto problémy jsou dobře patrné na předcházejícím obrázku (obr. 3.1). Oči a nalíčené rty již mají jinou barvu než tu pro kůži typickou. Naopak v pozadí obrázku se vyskytují většinou menší nespojité oblasti, které se jí podobají a operací prahování projdou.

Proto je potřeba použít tzv. morfologické operace. Jde o nelineární operace v obraze, kde každá operace je vnímána jako transofrmace mezi obrazem a tzv.

strukturním elementem. Ten je tvořen, stejně jako obraz, množinou bodů. Má však mnohem menší velikost, vetšinou jen 3x3 pixely. Nejčastější využití jsou operace pro předzpracování obrazu, při odstranění šumu, detekci hran apod. Pro detekci tváří nás však zajímají operace jiné, a to dilatace a eroze.

3.4.1 Dilatace

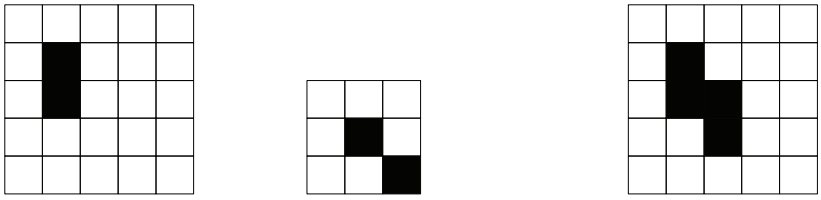
Jde o operaci, při které dochází k zaplňování malých děr nebo zálivů v obraze. Jejím efektem je prostě to, že zvětšuje objekty. Pokud tedy chceme docílit toho, aby velikosti objektů zůstaly stejné, musíme ji kombinovat s další operací – s erozí.

Operace dilatace sčítá dvě bodové množiny – vstupní obraz *A* a strukturní element *SE*. Tyto obrazy jsou nejčastěji binární, ale existují i šedotónové varianty.

Dilatace *D* je definována následovně:

$$D(A, SE) = A \oplus SE = \bigcup_{se \in SE} A_{se} \tag{3.8}$$

Rovnice popisující dilataci sice vypadá na první pohled složitě, ale její podstata je velmi jednoduchá. Dilataci lze brát jako srovnávání původního obrazu se strukturním elementem, který je po obraze postupně posouván. Pokud je střed (prostřední pixel) strukturního elementu shodný s aktuálním bodem obrazu pod ním (pokud jde o binární dilataci, tak v obou bodech je jednička), tak je strukturní element přičten k obrazu. Příklad dilatace je nakreslen na následujícím obrázku:



a) původní obrázek b) strukturní element c) výsledek dilatace

Obr. 3.2: Operace dilatace

3.4.2 Eroze

Jde o duální operaci k dilataci. Je definována jako průnik všech posunů *SE* s obrazem *A*. Efekt eroze je přesně opačný jako u dilatace – odstraňuje hranice objektů a

tím jej zmenšuje, zvětšují se díry a velmi malé objekty, které jsou menší jak strukturní element, zanikají. Díky postupnému zmenšování objektu se také používá k rozkládání na menší jednodušší části. V případě detekce tváří jej konkrétně můžeme využít k rozdělení krku a obličeje, více obličejů u sebe apod.

Definice vypadá takto:

$$E(A, SE) = A \ominus SE = \bigcap_{se \in SE} A_{-se} \tag{3.9}$$

Princip této oprace je v podstatě opačný dilataci. Strukturní element je postupně posouván po obraze a pokud není bod původního obrázku pod jeho středem roven jedné, je celý struktuní element od obrázku odečten. Z příkladu na předchozím obrázku (obr. 3.2) by tedy po erozi nezůstalo nic.

Abychom si mohli představit, jak taková detekce jen pomocí výběru barvy kůže a morfologických operací vypadá, použijeme obrázek 3.1 a popsané metody na něj aplikujeme. Nejprve se vyprahují všechny pixely nenáležící kůži dle rovnice 3.7 a vytvoří se binární maska – pixely kůže budou bílou barvou, ostatní černou. Výsledek těchto operací je na obr. 3.3.

Obr. 3.3: Binární maska obrázku

Z obrázku je patrný obrys tváře, jsou zde však okolo i chybně detekované místa. Na druhou stranu uprostřed tváře se v oblasti očí a úst nachází místa, která za kůži označena nebyla. Provedeme tedy nejprve operaci dilatace, aby se tato místa „zalila“ (obr. 3.4). Oblasti očí a úst pak již jsou označeny správně. Stejně tak ovšem se zvětšily chybně označené oblasti okolo. Proto je potřeba provést erozi, a to s větší intenzitou (více iteracemi) jako předchozí dilataci. Jejím výsledkem by mělo být to, že tyto osamocené oblasti zaniknou. Na obr. 3.5 již vidíme finální obrázek. Je zde již samotná tvář, bohužel i včetně vlasů (což

nemusí být až taková chyba a problém) a velmi malé části krku, která by například dalšími operacemi eroze mohla být odstraněna úplně.

Obr. 3.4: Binární maska po dilataci

Obr. 3.5: Výsledná binární maska

K příkladu na obrázcích je důležité poznamenat, že jde o dost ukázkový a ideální případ. Vyskytuje se zde pouze jedna tvář, navíc s pozadím poměrně výrazně odlišným, což dost pomáhá detekci. Mohou nastat i situace, kdy podobná metoda naprosto selže. V tom případě je třeba použít jinou, např. tu, která bude popsána v následující kapitole.

Jak taková chybná detekce vypadá je na následujících obrázcích:

Obr. 3.6: Příklad chybné detekce - zdrojový obrázek

Obr. 3.7: Binární maska chybné detekce kůže

Bohužel například právě barva dřeva je velmi podobná barvě lidské kůže, proto za ni byla omylem označena. Dále se na obrázku vyskytují jiné odkryté části těla (ruce) a jsou sice označeny správně jako kůže, ovšem detekci tváří to nijak neprospěje. Ani operace dilatace a eroze obrázku nepomůžou – je zde příliš mnoho chyb a tváře příliš blízko sebe, takže tato detekce selhává.

3.5 Výběr kandidátských oblastí

Kandidáty na obličej z vysledné binární masky obrázku (např. z obr. 3.5) vybereme již poměrně snadno. V případě složitějších obrázků s více tvářemi je vhodné provést ještě dále několikrát operaci eroze, abychom od sebe oddělili jednotlivé oblasti s tvářemi, v případě jedné tváře to již třeba není – záleží na konkrétní aplikaci detektoru, s jakými obrázky bude pracovat. Poté již u jednotlivých oblastí hledáme její okraje, srovnáváme velikost a podle toho se rozhoduje, zda by mohlo jít o tvář či nikoliv. V případě složitějších obrázků si ještě předtím můžeme pomoct malým trikem – jednotlivé oblasti si „vylijeme“ nějakou určenou (pokaždé jinou) barvou. Tím se nám každá oblast jednoznačně identifikuje, což nám při hledání jejich hranic může dost pomoci – budeme vědět, že nalezené místo v binární masce obrázku skutečně patří prohledávané oblasti a ne vedlejší, která do ní může být např. částečně vklíněna apod.

3.6 Lokalizace obličejových částí

Lokalizací jednotlivých částí obličeje, jako jsou oči, nos nebo ústa, můžeme ověřit správnost výběru kandidátských oblastí. Stejně tak ale tato metoda může být doplněna k ověření jiných druhů detekce (např. těch popsaných v kap. 4), případně ke zjištění biometrických údajů konkrétních osob.

Metod, kterými jednotlivé části detekovat, je více. Největší část z nich náleží mezi metody založené na zjevu, kde se provádí srovnávání obrazu s natrénovanými šablonami. Těmto metodám se ale věnuje až následující kapitola. Já se však pokusím popsat zajímavou metodu, která je založena na barevných transformacích obrazu, jejíž základy spočívají v různých odlišnostech obličejových částí oproti zbytku tváře. Tyto metody byly velmi dobře popsány v [8]. Všechny využívají barevného prostoru YCbCr a všechny se provádí jen na oblastech tváře a jen na oblastech kůže, ne tedy na celém obrázku s okolím či pozadím.

3.6.1 Lokalizace očí

Oči jsou jedním z nejvýraznějších rysů lidské tváře, proto je i jejich detekce poměrně jednoduchá a přesná. Jejich lokalizace má tři fáze. V první z nich se vytvoří mapa z chrominančních složek obrazu, v druhé pak z jasové složky. V poslední fázi se obě tyto složky zkombinují.

Nejprve je tedy nutné převést obrázek do barevného prostoru YCbCr, například dle rovnice 3.1. Všechny tři složky obrazu se poté normalizují na celý svůj rozsah, v případě 8-bitových hodnot je to tedy interval <0; 255>.

Poté se pro každý pixel obrazu vypočítačí tři různé hodnoty – Cb^2, $(255\text{-}Cr)^2$ a Cb/Cr. I tyto hodnoty se následně normalizují na plný rozsah. Poté se všechny sečtou, výsledek se vydělí třemi a vznikne nám hodnota *EyeMapC* – mapa očí z chrominančních složek. Celý vzorec vypadá takto:

$$EyeMapC = \frac{1}{3}(Cb^2 + (255 - Cr)^2 + Cb/Cr) \tag{3.10}$$

Myšlenka tohoto výpočtu je založena na znalosti, že oblast okolo očí má vysoké hodnoty složky Cb a naopak nízké hodnoty Cr, hlavně tedy v porovnání se zbytkem tváře.

Pomocí těchto operací je tedy možné tyto místa zvýraznit a extrahovat.

Druhá fáze pracuje jen s jasovou složkou obrazu. Je založena na skutečnosti, že lidské oko tvoří jak velmi světlé oblasti – bělmo, tak i velmi tmavé – zorničky. Aby tedy tato operace dávala dobré výsledky, je nutné, aby tyto dvě části oka byly na obrázku rozpoznatelné, aby obrázek neměl příliš malé rozlišení a nebyly tyto části zakryty (třeba mrknutím).

Opět se zde používají morfologické operace dilatace a eroze, tentokrát v šedotónové podobě a s kruhovým strukturním elementem. Nejprve se provede dilatace – tím se zajistí zalití oblasti zorniček a celkově se zvýrazní bělmo. Poté se s původním obrázkem provede eroze – tím se pro změnu zvětší oblast zorniček. Protože výsledek dilatace bude mít v oblastech očí nejvyšší hodnoty a výsledek eroze nejnižší, je ideální je podělit. Místa očí se tím velmi zvýrazní oproti ostatním částem obličeje a vznikne nám tak mapa nazývaná *EyeMapL*. Popsané operace jejího výpočtu formálně zapíšeme takto:

$$EyeMapL = \frac{Y(x,y) \oplus SE(x,y)}{Y(x,y) \ominus SE(x,y) + 1} \tag{3.11}$$

kde $\oplus$ označuje šedotónovou dilataci, $\ominus$ šedotónovou erozi se strukturním kruhovým elementem *SE*.

Nyní, když už máme obě mapy vypočítané, můžeme z nich spočítat výslednou mapu očí – *EyeMap*. Ta vznikne jednoduchým vynásobením obou map:

$$EyeMap = EyeMapC * EyeMapL \tag{3.12}$$

Vzniklou mapu je poté vhodné nějak upravit, ať už prahováním z ní odstranit nevyhovující místa (ty které nepatří očím) a případně opět provést morfologické operace.

Lokalizace očí na obrázku 3.1 s využitím binární masky z obr. 3.5:

Obr. 3.8: Příklad EyeMapC

Obr. 3.9: Příklad EyeMapL

Na obrázcích 3.8 a 3.9 jsou ukázky mapy očí. Čím světlejší je barva, tím větší je předpoklad, že se jedná o oči (s výjimkou bílého okolí tváře, které bylo již předem maskou vyloučeno). Následující dva obrázky pak už ukazují hotovou mapu očí *EyeMap* a její vyprahovanou variantu.

Obr. 3.10: EyeMap

Obr. 3.11: Vyprahovaná EyeMap

3.6.2 Lokalizace úst

Na podobném principu jako předchozí metoda funguje i lokalizace úst. Využívá přitom skutečnosti, že chrominanční složka Cr je na barvě úst daleko větší v porovnáním s Cb a také, že hodnoty Cr^2 jsou relativně mnohem vyšší neř Cr/Cb. Vzorec pro výpočet vypadá takto:

$$MouthMap = Cr^2 \cdot (Cr^2 - \eta \cdot Cr / Cb)^2 \tag{3.13}$$

Hodnotu η, což je jakási prahová hodnota, vypočteme pomocí vzorce 3.14:

$$\eta = 0{,}95 \cdot \frac{\frac{1}{n}\sum_{(x,y)\in FG} Cr(x,y)^2}{\frac{1}{n}\sum_{(x,y)\in FG} \frac{Cr(x,y)}{Cb(x,y)}} \tag{3.14}$$

Hodnoty Cr^2 a Cr/Cb by měly opět být normovány na celý rozsah hodnot <0; 255>, n je počet pixelů binární masky kůže FG (všechny výpočty jsou tedy, stejně jako u lokalizace očí, realizovány jen na pixelech náležících barvě kůže, resp. které jsou překryty binární maskou označující tvář s hodnotou 1). Na obr. 3.12 je příklad mapy MouthMap pro náš původní obrázek. Opět platí, že čím světlejší barva, tím větší pravděpodobnost úst.

Obr. 3.12: MouthMap

Kromě detekce očí a úst existují i způsoby detekce nosu – generování tzv. NoseMap. Je popsána např v [22] a využívá opět barevných transformací a morfologických operací k detekci nosních dírek. Jinou možností je použít hranový detektor k nalezení specifických tvarů nosu.

3.7 Využití detekce

Metody založené na invariantních rysech mohou často poskytovat velmi kvalitní výsledky, přitom jsou velmi jednoduché na implementaci. Dobrým příkladem použití může být detekce tváří a biometrických znaků např. na průkazových fotografiích – ty mají totiž správné osvětlení a ideální pozadí, u kterého nemůže dojít k záměně za tvář.

Často se tyto metody používají i jako doplňkové, u robustnějších algoritmů. Vyloučením všech barev neodpovídající kůži můžeme např. velmi zredukovat množství dat pro jiný následující způsob detekce. Stejně tak, díky své nenáročnosti, se nabízí implementace těchto metod v jednodušších zařízeních, jako jsou digitální fotoaparáty apod.

Nevýhodou těchto metod je už z principu jejich závislost na správném podání barev. Pokud máme třeba špatně vyváženou bílou barvu, vede to k posunu jednotlivých barevných složek a metody mohou selhat nebo produkovat daleko horší výsledky. Naprosto nepoužitelné jsou tyto metody v detekci na černobílých obrázcích. Jenže stejně jako lidský mozek dokáže rozeznat tvář i černobíle nebo třeba v úplně jiných barvách, tak musí existovat nějaká metoda využitelná na počítači, která to dokáže. Některé z nich budou popsány v následující kapitole.

4 METODY ZALOŽENÉ NA ZJEVU

Podstatou těchto metod by mohla být představa, že i člověk po narození nedokáže rozpoznávat svět a jako malé dítě se to musí naučit. A stejně jako se náš mozek trénuje a zdokonaluje v rozpoznávání svého okolí, tak je možno trénovat i počítače nebo stroje. Výkonnost lidského mozku je sice nesrovnatelně vyšší než toho nejlepšího procesoru na světě, ale i tak se za poslední roky v tato oblast posunula velmi dopředu a rozpoznávání tváří těmito metodami vykazuje velmi dobré výsledky.

4.1 Strojové učení

Anglicky „Machine learning“. Jde o základní stavební kámen těchto metod. Abychom mohli chtít po strojích, aby detekovaly tváře (nebo nějaké jíné objekty), musíme je nejdříve naučit co vlastně mají hledat. Proces učení probíhá většinou jednou – systém se nejprve naučí, poté jsou výsledky uloženy a před detekcí znova načteny. Během detekcí se už pak schopnosti nemění, systém se dál neučí (ale existují i metody, kde se může dále zdokonalovat). Podle toho, jak se systém učí, rozdělujeme tento proces na dva druhy – učení bez učitele a s učitelem.

4.1.1 Učení bez učitele

V tomto případě algoritmus nemá trénovací data a nezná proto ani požadované výstupní hodnoty – neví tedy, do jaké třídy data správně patří. Vstupy jsou proto shromažďovány a jsou považovány za náhodné. Z nich se pak na základě různých statistických postupů (technika extrakce charakteristických vlastností, Bayesova síť apod.) získávají různé statistické zákonitosti a na jejich základě se určuje výstup.

Vzhledem ke své složitosti a nejistým výsledkům se učení bez učitele používá jen zřídka. Ovšem je velmi podobné učení lidského mozku.

4.1.2 Učení s učitelem

Tato metoda je mnohem častější než předchozí. Algoritmu se během učení předkládají trénovací data (vstupní hodnoty) a k nim požadované hodnoty výstupu. Na

jejich základě se poté vytváří tzv. klasifikátor. Jde o funkci, která mapuje vstupní hodnoty na výstupní – příkladem vstupu může být nějaký obraz a výstupem logická 0 nebo 1, podle toho, jestli je na obraze natrénovaný hledaný objekt či nikoliv. Protože se během trénování používá jen omezená množina vzorů (není např. možné předložit algoritmu všechny existující lidské tváře na světě), je klasifikace vždy více či méně chybová. Snahou je tedy naučit jej tak a použít takové trénovací vzory, aby byla co nejmenší a pokryla co největší množství různých případů.

Během trénování se nejčastěji používají dvě různé sady vzorků – trénovací a testovací. Na trénovací se učí klasifikační funkce a na testovací se ověřuje její chyba. Je ovšem možné použít na obojí jen jednu sadu.

Před spuštěním trénování jsou ze vstupních dat vyextrahovány příznaky a vytvořen jejich vektor. Ten se během trénování nemění. Samotné trénování je pak opakující se proces, který se skládá z těchto dvou kroků:

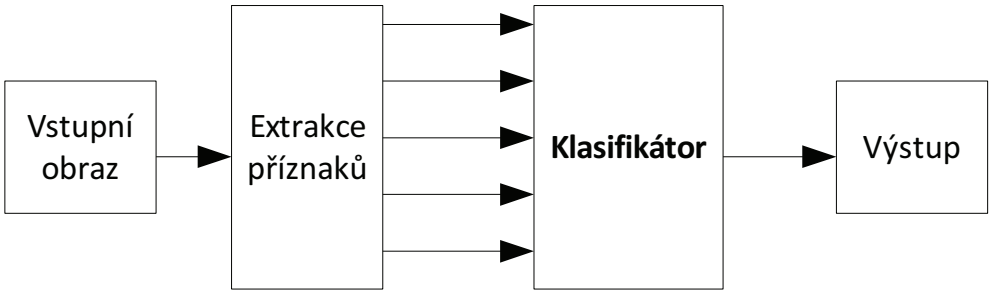
- Klasifikace vzorků – vektor příznaků se použije ke klasifikaci vzorků aktuální verzí klasifikátoru. Následně se pak zjistí jeho aktuální chyba.
- Úprava klasifikátoru – algoritmus se na základě chyby z předchozího kroku pokusí popupravit klasifikační funkci tak, aby byla chyba co nejmenší.

Celý proces trénování se opakuje do doby, než je splněno kritérium pro zastavení. Tím nejjednodušším je počet trénovacích kroků – kolikrát se proces opakuje je předem dáno. Dalším kritériem může být velikost chyby – trénování se opakuje do doby, dokud chyba neklesne pod určitou hranici. Nebo je možné testovat do doby, dokud se chyba mezi dvěma následujícími kroky snižuje o určitou mez. Poslední a asi nejlepší možností je použití již zmíněné testovací sady vzorků. V tomto případě se testuje do doby, dokud chyba klasifikace na této sadě klesá. V případě, že začíná stoupat, je již systém takzvaně přetrénován a je třeba učení zastavit – systém totiž ztrácí schopnost „generalizovat“ a začíná být naučen výhradně na trénovací vzorky a ne žádné jiné.

4.2 Klasifikátor

Jde o algoritmus, který je schopen úspěšně rozdělovat vstupní data s

charakteristickými atributy (příznaky) do předem zvolených výstupních tříd. Třídy mohou být dvě – v tom případě tuto klasifikaci nazýváme binární (jde o nejčastější případ), nebo může být tříd více.



Obr. 4.1: Blokové schéma systému s klasifikátorem

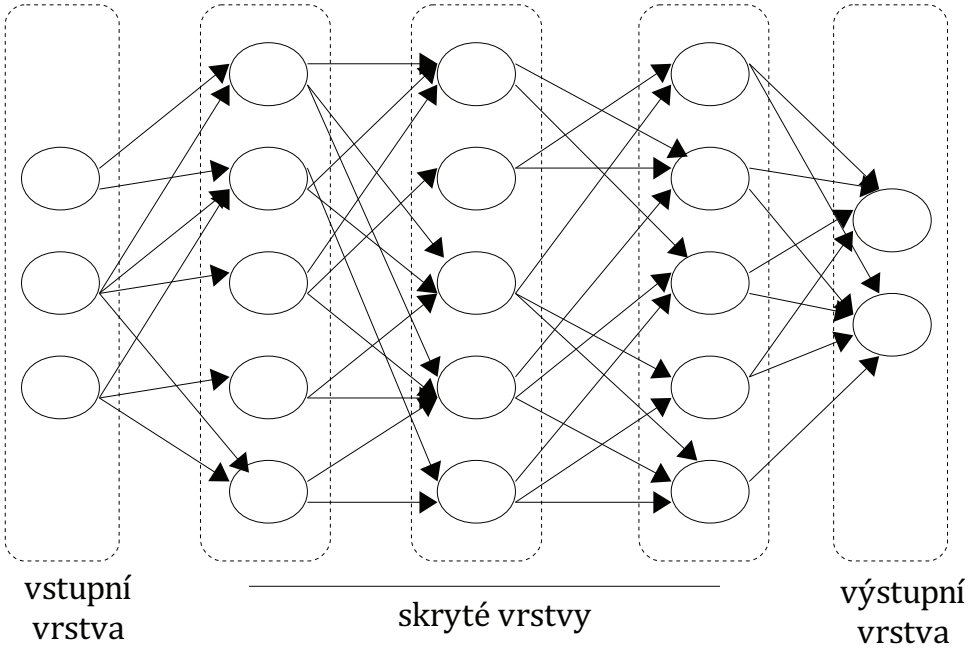
4.2.1 Neuronové sítě

Velmi často používaným klasifikátorem jsou umělé neuronové sítě. Jedná se o matematický model založený na sledování a pozorování skutečných biologických neuronových sítí. Ta se skládá z nervových buněk, neuronů, které umí pomocí elektrických vzruchů zpracovávat a přenášet informace.

Umělé neuronové sítě se skládají z několika mezi sebou vzájemně propojených vrstev. První se označuje jako vstupní, poslední jako výstupní a mezi nima se případně nachází tzv. skryté vrstvy. Podle způsobu propojení mezi vrstvami se poté rozlišuje několik typů sítí:

- Dopředná síť – vyznačuje se tím, že v ní neexistují smyčky a informace tak proudí přímo od vstupů k výstupům. Výstupy z předchozí vrstvy vedou jen do vrstvy následující a žádné jiné. Nejjednodušším typem této sítě je jednovrstvý perceptron – ten se skládá jen z jedné (výstupní) vrstvy neuronů. Díky své jednoduchosti však nedokáže vyřešit některé jednoduché úkoly, například XOR funkci. Ta je řešitelna až s více vrstvami.

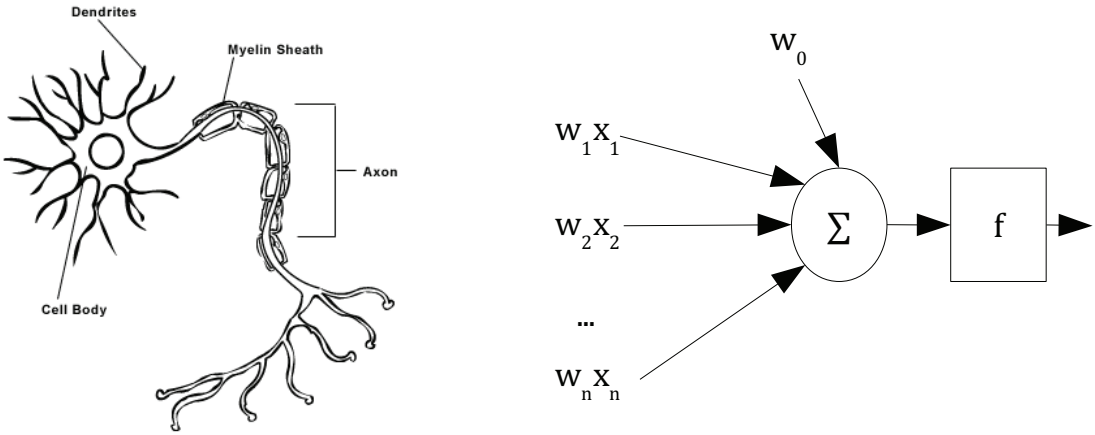
- Rekurentní síť – v tomto typu sítě se již vyskutují smyčky a zpětné vazby, data tedy již neproudí jen přímo od vstupu k výstupům. Typickým příkladem je Hopfieldova síť. Ta obsahuje skupinu neuronů, které jsou vzájemně každý s každám propojeny. Někdy se tomuto propojení říká „asociativné paměť“.



Obr. 4.2: Dopředná neuronová síť

4.2.2 Neuron

Stejně jako biologický neuron je i ten umělý tvořen mnoha vstupy (dendrity), jádrem a jedním jediným výstupem (axonem). Umělý neuron má tedy *n* různých vstupů označených $x_1, x_2, \ldots, x_n$, ke každému vstupu je přiřazena jeho váha *w.* Kromě toho je na vstupu i speciální váha w_0, která ovlivňuje práh rozhodování. Jádro neuronu pak provádí součet všech vážených vstupních hodnot. Na výstupu je pak určitá přenosová funkce – ta může mít různé průběhy, např. skokový, sigmoidální apod., v závislosti na typu a požadavcích úlohy.

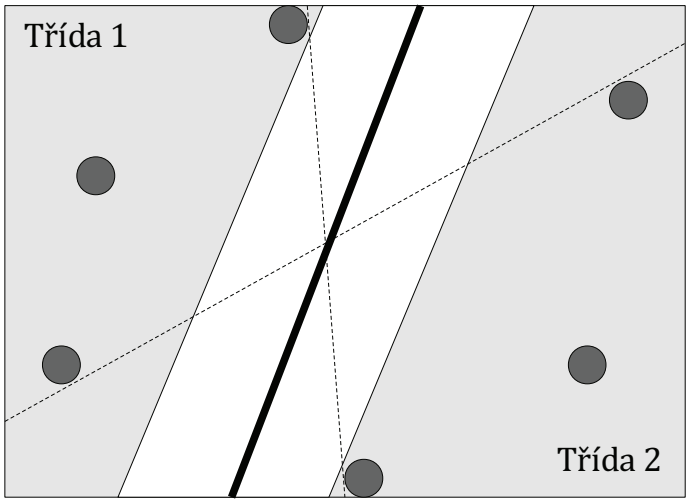


Obr. 4.3: Skutečný a umělý neuron

Proces trénování neuronové sítě je vlastně jen nastavování jednotlivých vah neuronů tak, aby síť vracela co nejlepší výsledky podle trénovací množiny. Asi nejznámější metodou trénování sítě je Back-propagation.

4.2.3 Support Vector Machine

Další možnou metodou klasifikace je vedle neuronových sítí Support vector machine – SVM [14, 21]. Jde o metodu klasifikace lineárních dat, která se při vyhledávání obličejů často používá. Proces trénování klasifikátorů se za pomoci trénovacích dat snaží najít hyperplochu co nejlépe rozdělující data v prostoru. Na následujícím obrázku jsou data dvou tříd, čárkovanými čarami jsou hyperplochy s nejmenší dělící vzdáleností, plnou čarou hyperplocha, která dělí nejlépe.



Obr. 4.4: Rozdělení prostoru na hyperplochy

Taková data, které není možné lineárně (pomocí přímek) rozdělit, je třeba nejprve transformovat do prostoru s vyšší dimenzí pomocí nějakého jádra (kernelu).

4.2.4 Principal Component Analysis

Tato technika (zkráceně PCA, v překladu Analýza hlavních komponent) je používána ke snížení dimenze dat s do nejmenší ztrátou informací v nich obsažených. Je někdy označována jako Htellingova nebo Karhunen-Loeveho transformace. Cílem je redukce původního počtu proměnných novými umělými veličinami, zde nazývanými komponenty. Ty shrnují informaci o původních proměnných za cenu co nejmenší ztráty informace. Veličiny jsou na sobě nezávislé a jsou určitým způsobem seřazeny.

Samotný princip pak spočívá v určení kovarianční matice a jejích vlastních čísel a vektorů – komponenty. Číslo s nevyšší hodnotou se nazývá hlavní komponenta (principal component). Velikost ostatních vlastních čísel pak označuje jejich důležitost. Zanedbáním těch s nejmenší hodnotou se dosahuje omezením dimenze, přitom však s nejmenší ztrátou informace.

4.3 Obrazové příznaky

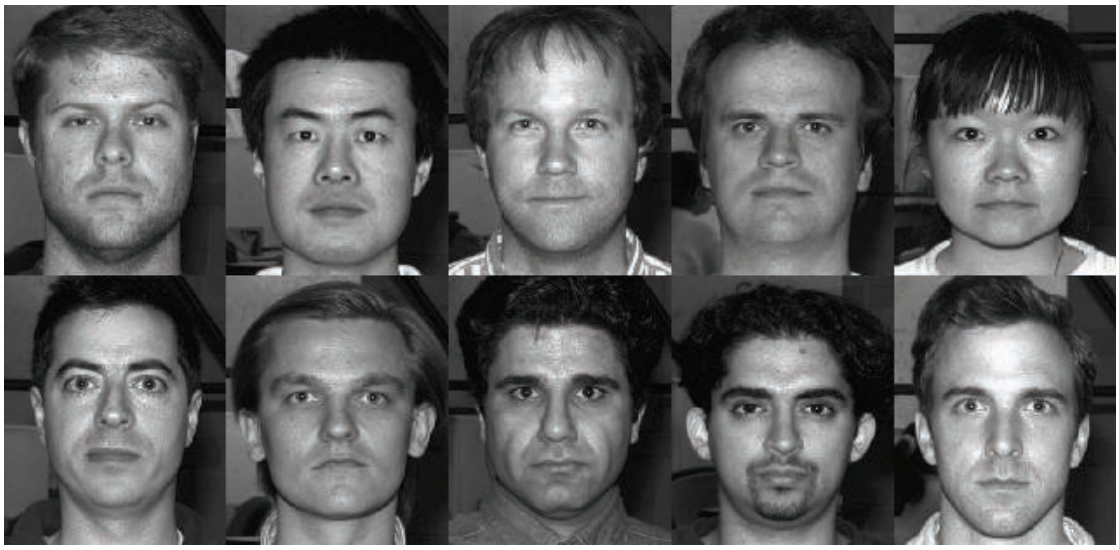
V předchozích kapitolách je vysvětleno, co je to klasifikátor, jakými druhy učení jej můžeme trénovat, ale pořád je potřeba vysvětlit, co je naprostým základem informace získávané z obrazu – obrazovým příznakem. Jde ve většině případů o nějakou číselnou hodnotu určitým způsobem získanou z obrazu nebo jeho části – často konvoluci obrazu na určitém místě o určité velikosti s nějakým daným konvolučním jádrem. Nejčastějším typem jsou tzv. Haarovy příznaky vycházející z Haarových vlnek, těm se budu podrobněji věnovat v kapitole 5, kde popisuji implementaci detektoru v OpenCV. Kromě Haarových příznaků se někdy používají třeba Gáborovy vlnky.

4.4 Databáze tváří

Abychom mohli vůbec učení algoritmu uskutečnit, je potřeba opatřit si vhodnou trénovací a testovací sadu snímků. Vytvoření vlastních snímků je velmi pracné a časově náročné, proto je lepší použít nějakou už existující. Těch je celá řada, ať už placených nebo volně dostupných. Liší se také v počtu osob, snímků, jestli jsou tváře nasnímány jen zepředu nebo z různých póz atd. V následující tabulce jsou uvedeny nejznámější veřejně dostupné databáze [9].

Obr. 4.5: Snímání 3D povrchu tváří pro databázi MIT

Databáze	Subjektů	Rozlišení	Obrázků	Popis
AR	116	768x576	3288	obsahuje různé výrazy osob, pózy, nasvětlení, předměty zakrývající tvář
BANCA	208	720x576	-	každá osoba byla 12x nasnímána během tří měsíců
CAS-PEAL	1040	360x480	30900	7 kategorií fotek, 9 kamer okolo subjektu, které pořídí 27 fotek při natočení hlavy rovně, nahoru a dolů
CMU Hyperspectral	54	640x480	-	fotografie v různém barevném spektru – od viditelného po IR
CMU PIE	68	640x480	41368	různé výrazy, pózy, nasvětlení
Equinox IR	91	240x320	-	čtyřsekundové sekvence v IR spektru
FERET	1199	256x384	14051	24 kategorií, obličeje násnímány z úhlu -60° až +60°
Korean Face DB	1000	640x480	52000	modré pozadí, různé výrazy, pózy, nasvětlení, úhly pohledu
MIT	200	256x256	-	3D data nasnímané laserovým scannerem
Notre Dame HID	> 300	1600x1200	> 15000	subjekty opakovaně snímány s 2-týdenní periodou
U. Texas	284	720x480	-	4 různé kategorie, snímáno jako videosekvence
Yale B	10	640x480	5850	64 různých světelných podmínek, 9 různých póz



Obr. 4.6: Ukázka tváří z Yale B databáze

5 DETEKCE OBLIČEJŮ POMOCÍ OPENCV

5.1 OpenCV

OpenCV (Computer Vision) je knihovna funkcí pro práci s grafikou. Její vývoj začala firma Intel v roce 1999. Jejím hlavním cílem bylo vytvořit prostředí, které bude otevřené, velmi rychlé i pro real-time zpracování grafiky, jednoduché k použití a také bez nějakých omezujících licencí. Je napsána v jazyce C, což umožňuje její portování pro celou řadu procesorů jako například různé DSP apod. Jako licence byla zvolena BSD – to umožňuje použití i v komerčních projektech bez nutnosti zveřejňovat jejich zdrojové kódy. Ovšem všechny zdrojové kódy samotného OpenCV, jak už název napovídá, jsou volně k dispozici.

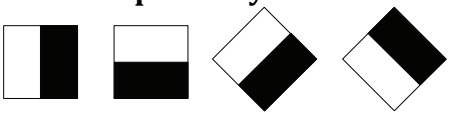
5.2 Detekce objektů v OpenCV

Detektor obsažený v OpenCV patří do skupiny detektorů založených na strojovém učení. Autory algoritmu, který používá, jsou Viola a Jones [17], kteří jej publikovali v roce 2001. Tento algoritmus byl dále ještě vylepšen Rainerem Lienhartem [10, 11]. Fázi učení tedy zajišťuje algoritmus AdaBoost a k výpočtu hodnot příznaků využívá Haarovy vlnky.

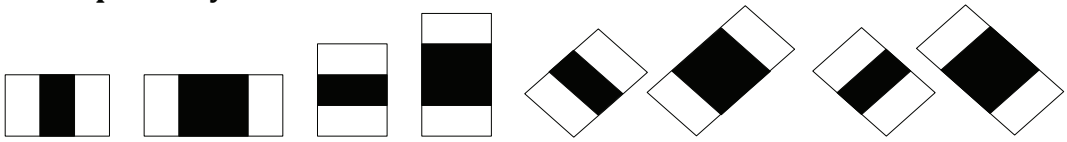
5.2.1 Typy Haarových příznaků

Jde o příznaky podobné Haarovým vlnkám. Existuje jich několik druhů, OpenCV podporuje následující [13]:

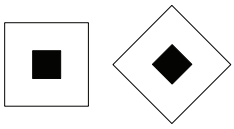
- **Hranové příznaky**



- **Čárové příznaky**



- **Středové příznaky**



Obr. 5.1: Typy Haarových příznaků

Nejvíce využívané jsou vodorovné a svislé hranové příznaky, vodorovné a svislé čárové a základní neotočený středový příznak. Je to hlavně z toho důvodu, že jsou nejjednodušší a snadno se dá vypočítat jejich hodnota.

5.2.2 Výpočet hodnot příznaků

Způsob, jakým se počítají hodnoty příznaků, je poměrně jednoduchý. Příznaky se jakoby přiloží v určitém místě a velikosti na obraz, vypočítá se suma všech pixelů pod bílou částí, suma pod černou částí a obě tyto hodnoty se pak od sebe odečtou – tím získáme hledanou hodnotu celého příznaku.

Jak už jsem naznačil, příznaky nejsou počítány pro obraz jako celek, ale vždy jen na nějaké jeho části – tzv. podokně. Toto podokno se jednak posouvá po obraze s určitým krokem (nejčastěji 2 pixely) a jednak také mění svou velikost. Díky tomu je těchto podoken obrovské množství – třeba pro relativně malý obrázek o rozlišení 320x240 bodů jich může být okolo půl miliónu.

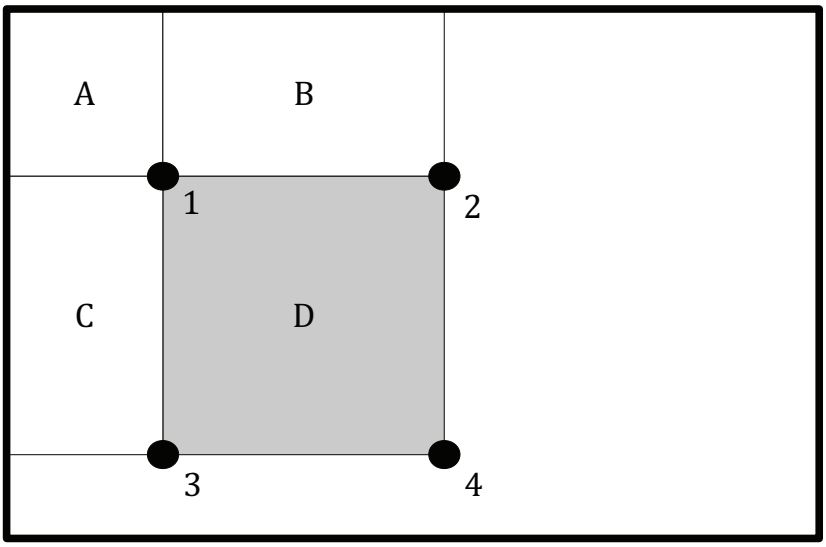
Počítání tak obrovského množství hodnot příznaků by bylo extrémně pomalé a neefektivní (máme statisíce až milióny podoken a v každém z nich ještě musíme procházet všechny pixely a sčítat je), proto se místo zdrojového obrazu používá takzvaný integrální obraz. Zde jednotlivé body nepředstavují přímo hodnotu daného pixelu, ale součet aktuálního a všech předchozích pixelů. Musí zde platit tyto vztahy [15]:

$$s(x, y) = s(x, y-1) + I(x, y) \tag{5.1}$$

$$I_{\text{int}}(x, y) = I_{\text{int}}(x-1, y) + s(x, y) \tag{5.2}$$

kde s(x, y) je kumulovaný součet hodnot v řádku, I(x, y) je původní obraz a I_{int}(x, y) obraz integrální

Výpočet hodnot příznaku (což je v podstatě součet hodnot bodů obrazu ve vymezeném obdelníku) se díky integrálnímu obrazu velmi zjednoduší. Z mnoha stovek nebo tisíc operací jen na dvě sčítání a jedno odečítání.



Obr. 5.2: Výpočet podokna pomocí integrálního obrazu

Pokud tedy chceme vypočítat součet pixelů v obdelníku D z obrázku 5.2, odečteme od sebe hodnoty z bodů 4 a 3, pak z bodů 2 a 1 a oba výsledky opět odečtem. Po úpravě tohoto vztahu dostaneme symbolickou rovnici:

$$\sum\nolimits_D = (I_{\text{int}}(4) - I_{\text{int}}(3)) - (I_{\text{int}}(2) - I_{\text{int}}(1))$$

$$\sum\nolimits_D = (I_{\text{int}}(4) + I_{\text{int}}(1)) - (I_{\text{int}}(2) + I_{\text{int}}(3)) \qquad (5.3)$$

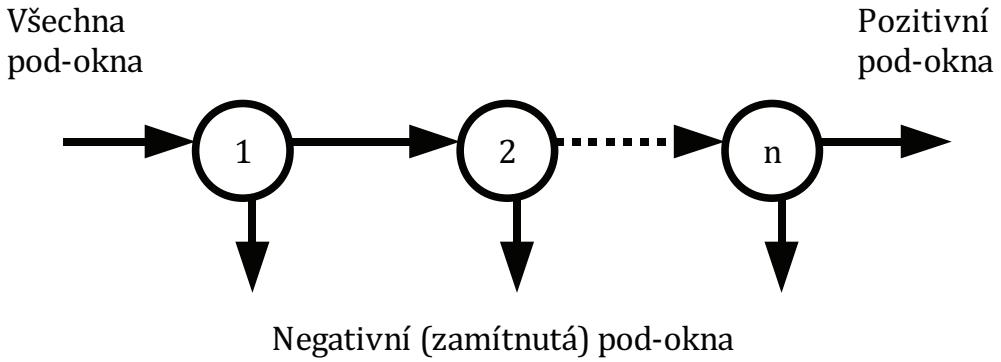
kde $I_{int}(x)$ představuje hodnotu integrálního obrazu v jednotlivých bodech dle obr. 5.2

5.2.3 Kaskádové zapojení klasifikátorů

V předchozí kapitole je popsáno, jakým způsobem se počítá hodnota obrazového příznaku. Ale i když si jejich výpočet velmi usnadníme použitím integrálního obrazu, stále je třeba zkoumat obrovské množství pod-oken. Abychom se tomu vyhnuli a co nejdříve došli k výsledku, používá se tzv. kaskádové zapojení. Princip vychází z toho, že většinu všech obrazů tvoří nežádoucí pozadí a tváře (případně jiné hledané vzory a objekty) zabírají jen menší část. Další důležitý poznatek je, že k relativně spolehlivému

rozlišení, zda je v obraze obličej nebo není, stačí jen pár vhodně vybraných příznaků.

Proto se nejdříve detektor snaží zamítnout co nejvíce podoken bez tváře (negativní) a dál kaskádou pokračují jen okna označená jako pozitivní.



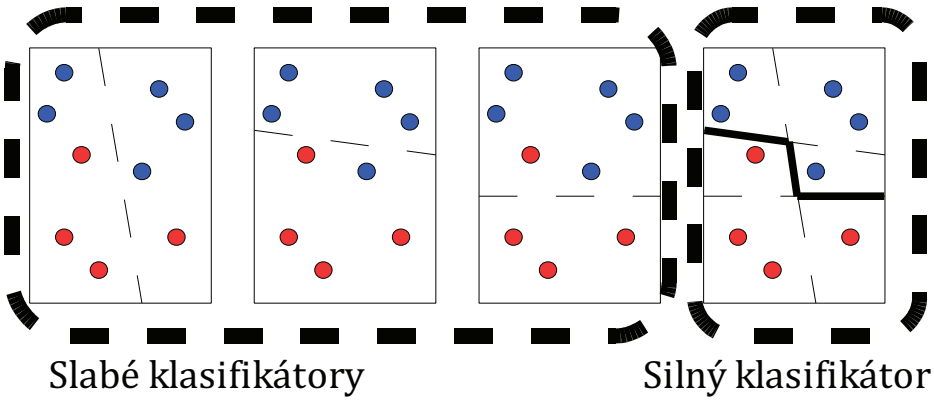
Obr. 5.3: Kaskádové zapojení klasifikátorů

Kaskáda na obrázku 5.3 má celkem *n* stupňů. Každý stupeň (což je vlastně klasifikátor, který rozhoduje, zda se jedná nebo nejedná o hledaný vzor) je trénován zvlášť. Nejdříve obsahuje jen jeden příznak a postupně se přidávají další, dokud není dosaženo určité požadované hranice ve schopnostech detekce.

5.3 Učení klasifikátorů

Aby nám jednotlivé klasifikátory správně rozlišovaly co je a není tvář, je potřeba je to naučit. K tomu účelu je v OpenCV implementován algoritmus strojového učení zvaný AdaBoost [1, 2], konkrétně jeho varianty Discrete AdaBoost, Real Adaboost, Gentle AdaBoost a Logitboost [13]. Hlavním úkolem těchto algoritmů je vybrat z obrovské množiny příznaků ty nejvhodnější a vytvořit z nich fungující klasifikátor. Trénování samotné se provádí na sadách testovacích obrazů, u kterých předem víme, zda se jedná o pozitivní (je na nich hledaný vzor nebo tvář) nebo negativní (cokoliv jiného).

Máme tedy množinu všech tzv. slabých klasifikátorů $h(x_i)$, což jsou v podstatě funkce, které na základě hodnoty svého (jednoho jediného) příznaku a daného prahu vrací -1 nebo 1. Někdy se tyto klasifikátory označují jako „slabí žáci“ (weak learners). Tento klasifikátor může být tvořen například i neuronem nebo neuronovou sítí popsanou v kap. 4.2.1.



Obr. 5.4: Slabé a silné klasifikátory

Přesnost těchto slabých klasifikátorů nemusí být o moc větší než 50%. Důležitou věcí je však to, že k těmto klasifikátorům se postupně přidávají další s podobnou přesností, čímž vznikne soubor slabých žáků zvaný „silný žák“ (strong learner) *H(x)*. Hodnota tohoto silného žáka se poté vypočítá jakou součet všech slabých žáků v něm obsažených, každý vynásoben svou vahou *w*.

$$H(x)=\sum^{i} h(x_i)\cdot w(x_i) \tag{5.4}$$

Právě váhy jednotlivých slabých žáků je to, co potřebujeme během učení nastavit. Ze začátku jsou inicializovány rovnoměrně (na rozdíl od některých jiných algoritmů například při učení neuronových sítí, kde jsou nastaveny náhodně). V každém kroku se vybere slabý žák, který vykazuje nejmenší chybu. Podle něj se pak nastaví váhy ostatních tak, aby žáci s největší chybou měli největší váhu. Poté se nechají klasifikátory znova detekovat na trénovací množině, opět se vybere slabý žák, upraví váhy a tak pořád dokola. Díky tomuto postupu redukuje AdaBoost chybu trénování exponenciálně v závislosti s rostoucím počtem klasifikátorů [15].

Občas ale zvyšování počtu klasifikátorů vede k takzvanému přetrénování. V tuto chvíli začínají být klasifikátory naučeny příliš moc na trénovací množinu a ztráci schopnost rozpoznávat i jiné, podobné – velmi správně a přesně například detekují obrazy z trénovací množiny nebo velmi podobné, jiné nové tváře (vzory) už však ne. Pokud jsme se s učením do této fáze dostali, je třeba jej ukončit. Naštěstí se tohle stává jen zřídkakdy a dost to souvisí s výběrem správné trénovací množiny a dostatečným počtem trénovacích vzorků.

Algoritmus AdaBoost dle [2]

Vstup:

Dvojice (x_i, y_i) – testovací obraz a odpovídající hodnota -1 nebo +1 podle toho, zda se jedná či nejedná o hledaný vzor:

$(x_1, y_1), \ldots, (x_n, y_n)$ kde $x_i \in X, y_i \in Y = \{-1, +1\}$

Rovnoměrná inicializace vah D_1:

$$D_1(i) = \frac{1}{m}, \quad i = 1, \ldots, m$$

A nakonec zvolíme počet iterací T

Cyklus pro t = 1, … , T :

Vypočítáme vážené trénovací chyby:

$$\epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$$

Poté vybereme klasifikátor s nejmenší chybou

$$h_t = arg \min_{h_j \in H} \epsilon_j$$

Ověříme, zda $\varepsilon_t < 0{,}5$ jinak ukončíme trénování

Nastavení α_t na zvolenou hodnotu, nejčastěji vypočtenou takto:

$$\alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t}$$

Následná úprava vah:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

kde Z_t je normalizační faktor:

$$Z_t = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

Výstup:

Výpočet finálního klasifikátoru:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x) \right)$$

5.4 Učení pomocí funkcí OpenCV

Protože je problematika strojového učení velmi rozsáhlá a napsat rychlý a opravdu kvalitní trénovací algoritmus není zrovna jednoduché, je nejvhodnější použít již připravený program, který je přímo součástí základní instalace OpenCV – utilita *HaarTraining*. S její pomocí lze vytvořit soubor natrénovaných kaskád klasifikátorů, které můžeme použít nejenom k detekci tváří, ale v podstatě jakéhokoliv vzoru - záleží jen na tom, jaké obrázky mu budeme během trénování předkládat. V případě tváří bohužel máme tu nevýhodu, že kvalita natrénování dost závisí na množství předložených trénovacích vzorků – ve většině případů totiž platí, že čím více, tím lépe. Je tedy vhodné opatřit si třeba sadu trénovacích vzorků z nějaké databáze vyjmenované v kapitole 4.4.

5.4.1 Příprava vzorků k trénování

K trénování potřebujeme dvě různé sady vzorků – negativní a pozitivní. Příprava negativních vzorků není tak složitá – obsahem těchto vzorků jsou části pozadí apod., v žádném případě by se v nich neměla objevit žádná tvář. Jejich seznam je pak uložen v tzv. souboru pozadí, což je obyčejný textový soubor s cestami k jednotlivým obrázkům. Podobně jako negativní vzorky jsou uloženy i ty pozitivní. Jejich seznam je také v textovém souboru, kromě cest je zde však ještě definován obdelník, kde přesně se tvář nachází. Je možné na jednom vzorku takto označit i více tváří najednou.

5.4.2 Spuštění trénování

Trénovací program se spouští z příkazové řádky. Má několik možných parametrů, kterými jednak definujeme zdrojové a výsledné soubory s daty a jednak můžeme ovlivnit kvalitu a rychlost trénování. Jejich popis je na následující straně.

Přehled parametrů programu HaarTraining

–data <název_adresáře>

určuje adresář, do kterého se ukládají výsledné soubory s natrénovanými klasifikátory

–vec <nazev_souboru>

název textového souboru se seznamem pozitivních vzorků

–bg <nazev_souboru>

název souboru s negativními vzorky

–npos <pocet_pozitivnich_vzorku>

počet pozitivních vzorků

–nneg <pocet_negativnich_vzorku>

počet negativních vzorků

–nstages <pocet_stupnu_kaskady>

počet stupňů vytvořené kaskády

–mem <velikost_v_MB>

velikost pomocné paměti – čím víc, tím rychlejší trénování bude

–minhitrate <pocet>

minimální počet správných detekcí pro každý stupeň kaskády

–maxfalsealarm <pocet>

maximální počet chybných detekcí pro každý stupeň

–weighttrimming <pomer>

poměr, jakým se zmenšuje podokno při vytváření příznaků

–mode <BASIC | CORE | ALL>

vybírá typ haarových příznaků: BASIC označuje jen jednoduché pravoúhlé, ALL označuje všechny typy včetně jejich 45° rotací

–w <velikost>

–h <velikost>

velikost trénovacích obrázků

5.4.3 Soubory s natrénovanými klasifikátory

Pokud se blíže podíváme na vytvořené soubory, které vznikly trénováním, tak zjistíme, že jde o .xml soubor s přehlednou strukturou. Kořenovým elementem je zde <*Cascade*>, pod ním pak jednotlivé stupně označené <*Stages*> a každý z nich pak obsahuje určitý počet definovaných příznaků <*Features*>.

Pokud by se nám nechtělo trénovat si vlastní klasifikátory, obsahuje OpenCV již sadu předem připravených. Jde o čtyři různé soubory pro detekci tváře zepředu a po jednom souboru pro detekci tváře z profilu, pro detekci očí bez brýlí a s brýlemi, celého těla nebo jeho horní a dolní půlky. Tyto soubory byly natrénovány samotnými tvůrci těchto algoritmů a poskytují poměrně velmi dobré výsledky (viz kap. 6.3).

5.5 Detekce pomocí natrénovaných souborů

Pokud už máme natrénovaný soubor s klasifikátory, můžeme jej konečně použít k detekci. Jde už asi o nejsnadnější část, pokud se tedy nepustíme do vytváření vlastního detektoru a nepoužijeme ten, který máme k dispozici v OpenCV. To už obsahuje jak fuknce pro samotné načtení souboru, tak funkce detektoru podle Viola-Jones.

5.5.1 Načtení souboru příznaků

K načtení souboru si můžeme vybrat ze dvou funkcí. První z nich je *CvLoad()*, což je univerzální funkce k načítání objektů z .xml souborů. Druhou možností je použít fuknci *CvLoadHaarClassifierCascade()*, která je přímo určena k načítání souborů a objektů vytvořených podle kap. 5.4. Ovšem lze s klidným svědomím použít obě funkce.

5.5.2 Detekce podle souboru

Když máme soubory načteny, můžeme se pustit do detekce. Ta je realizována funkcí *cvHaarDetectObjects()*. Má několik důležitých parametrů, které ovlivní celou detekci. Kromě klasických, jako je určení obrázku, na kterém detekujem a ukazatele na načtený soubor kaskád je to *scale_factor* a *min_neighbors*.

scale_factor – jde o poměr, jakým se postupně zmenšuje podokno při prohledávání obrazu (více o podoknech v kapitole 5.2.2). Musí být vždy vetší než 1,

většinou se nastavuje na 1,1. Pokud tento poměr nastavíme příliš velký, může teoreticky nastat situace, že obličej velikostí „mineme“ - v jednom průchodu bude okno příliš veliké a tvář v něm nepůjde rozpoznat, v následujícím průchodu už bude pro změnu okno tak malé, že se tvář do něj nevejde. Pokud však ale nastavíme poměr zbytečně malý, velmi si tím můžeme prodloužit čas detekování.

min_neighbors – v předchozím odstavci jsem se zmiňoval o tom, že můžeme tvář při hledání minout. Může však nastat i opačná situace (a ta také nastává a to téměř vždy), a to že jedna tvář bude detekována vícenásobně. Jak se podokno při hledání posunuje a mění svoji velikost, nachází tvář opakovaně – pokaždé s trochu jinou pozicí. Tento jev je třeba nějak potlačit a sloučit nalezené oblasti do jedné. Zároveň ho však můžeme využít a za tváře prohlásit jen ty oblasti, na kterých jsme je detekovali vícekrát – a právě kolikrát, to nám určuje tento parametr. Může být nastaven na 1, v tom případě ale produkuje poměrně hodně chybně pozitivních detekcí.

6 PRAKTICKÁ REALIZACE DETEKTORU

K naprogramování detektoru jsem si zvolil vývojové prostředí MS Visual Studio 2008 a samozřejmě jsem využil služeb knihoven OpenCV. Mým cílem nebylo ani tak vytvořit co nejlepší detektor tváří, ale spíše vyzkoušet si některé metody, ověřit si funkčnost a to jak pracují.

6.1 Detekce barvy kůže

První věcí, které jsem se věnoval, bylo rozeznávání barvy kůže. Zkoumal jsem rozdíly separovatelnosti těchto barev v různých barevných modelech, konkrétně v RGB, normalizovaném RGB, HSL a YCbCr. Jako nejlepší se nakonec ukázal model YCbCr, jednak proto, že má oddělenou jasovou složku (to má ale i HSL) a jednak také pak lze využít složek Cb a Cr a jejich vzájemných závislostí a poměrů při detekci částí tváře.

Jak dopadlo rozlišení barvy kůže v různých barevných modelech (viz kap. 3.3) a na různých příkladech obrázků vidíte zde:

Obr. 6.1: Zdrojové obrázky pro detekci kůže

Obr. 6.2: Výběr barvy kůže z prvního obrázku v RGB prostoru a v YCbCr

Obr. 6.3: Výběr barvy kůže z druhého obrázku v RGB prostoru a v YCbCr

Detekce kůže na obrázku s velmi slabým osvětlením – srovnání RGB modelu, který nemá oddělenou jasovou složku s YCbCr:

Obr. 6.4: Originál málo osvětleného obrázku

Obr. 6.5: Detekce v RGB modelu

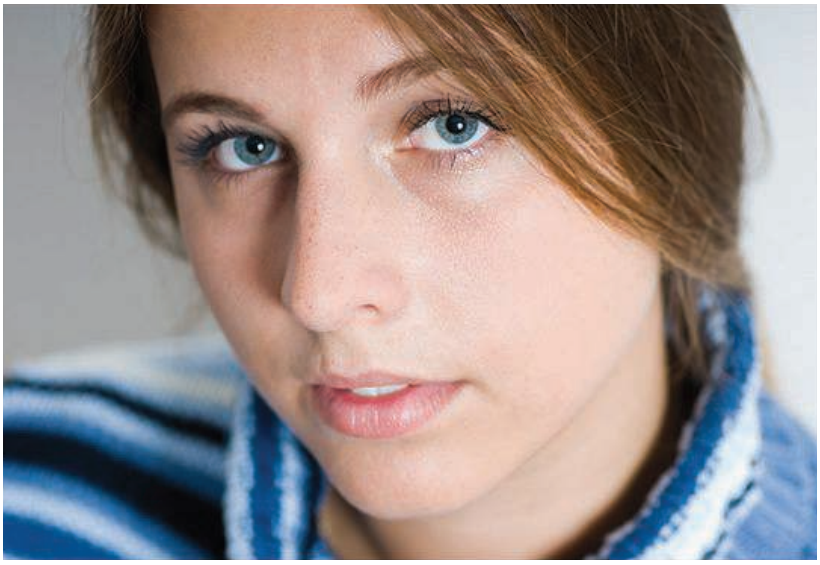
Obr. 6.6: Detekce v YCbCr modelu

Jak je vidět na předchozích obrázcích, vyskytují se v obraze chybné detekce – je detekováno i to co kůže není (většinou malé a nespojité oblasti, hlavně na obr. 6.3) nebo naopak není správně detekována část obličeje na obr. 6.2 u YCbCr barevného modelu. Také oči, ústa nebo nos do barevného prostoru kůže nepatří, přitom do obličeje ano. Proto přichází čas na morfologické operace (viz kap. 3.4). Různými pokusy a testováním jsem zjistil, že jako první je lepší provést dilataci. Tím se zaplní „díry“ v obličeji, jako třeba právě oči. Ovšem dilatace vede ke zvětšování objektů v obraze – to má za následek i zvětšení objektů nežádoucích. Proto následuje eroze. Jestliže jsme provedli dilataci např. s pěti iteracemi (hranice objektů se zvětší o 5 pixelů v případě použití jednoduchého čtvercového 3x3 strukturního elementu), musíme erozí provést taky nejméně s pěti iteracemi, plus ještě dalšími, abychom původní nechtěné objekty odstranili úplně.

S kolika iteracemi přesně morfologické operace provádíme záleží na charakteru předkládaných obrázků – na jejich rozlišení a také tomu, jak podstatnou část z nich tvoří hledané tváře. Nevhodně provedenou dilarací nebo erozí můžeme detekci i dost uškodit.

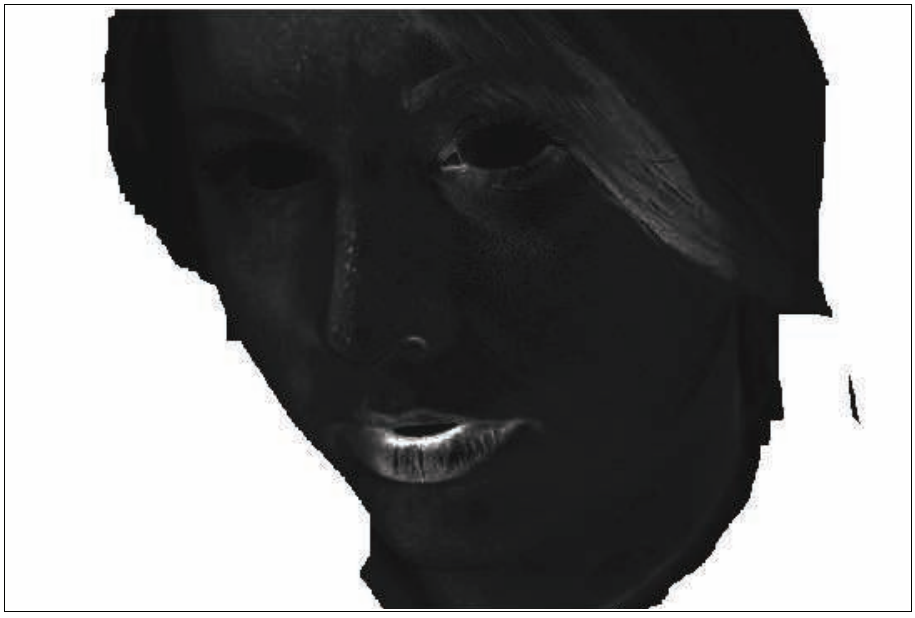
6.2 Lokalizace obličejových částí pomocí barev

Naprogramoval jsem algoritmy pro detekci očí a úst dle kapitoly 3.6. Tyto metody detekce jsou ovšem použitelné jen na některých typech obrázků, jako jsou portréty nebo průkazové foto. Na obr. 6.2 a 6.3 jsou naprosto nepoužitelné – vyžadují totiž, aby šlo jen o pixely patřící tváři a ne vlasům, rukám, nohám apod.



Obr. 6.7: Portrét pro detekci očí a úst

Obr. 6.8: EyeMap - mapa očí



Obr. 6.9: MouthMap - mapa úst

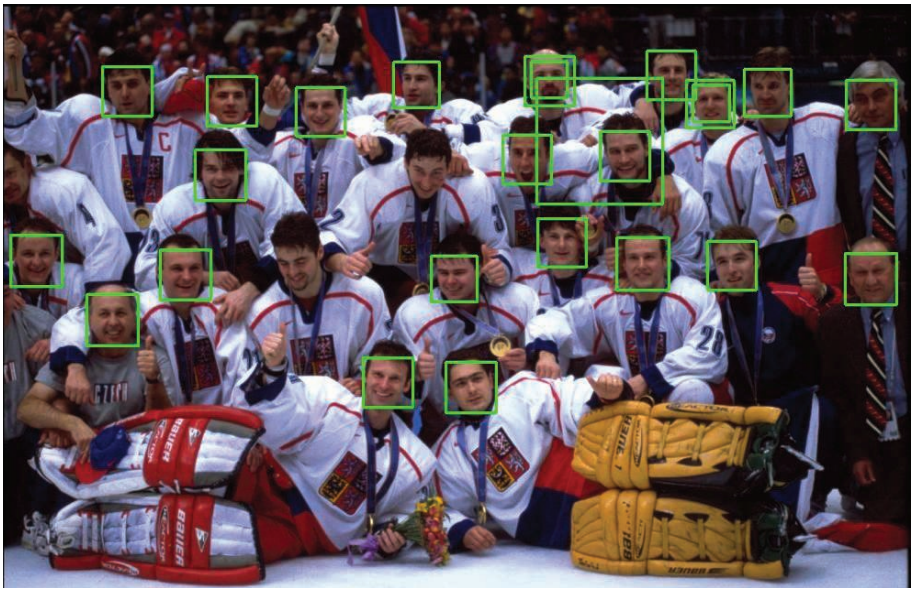
Pokud máme k dispozici vhodný obrázek, jako je např. obr. 6.7, dopadá detekce očí a úst velmi dobře. Pokud bychom detekované mapy morfologickými operacemi ještě upravili a vyprahovali, můžeme pozici těchto částí obličeje přesně lokalizovat. Navíc jejich existence a vůbec geometrie je vhodným způsobem, jak ověřit, že jsme detekovali tvář a ne chybně něco jiného.

Ukázka zdrojového kódu těchto detekcí je v příloze na konci práce.

6.3 Detekce obličejů Haarovými příznaky

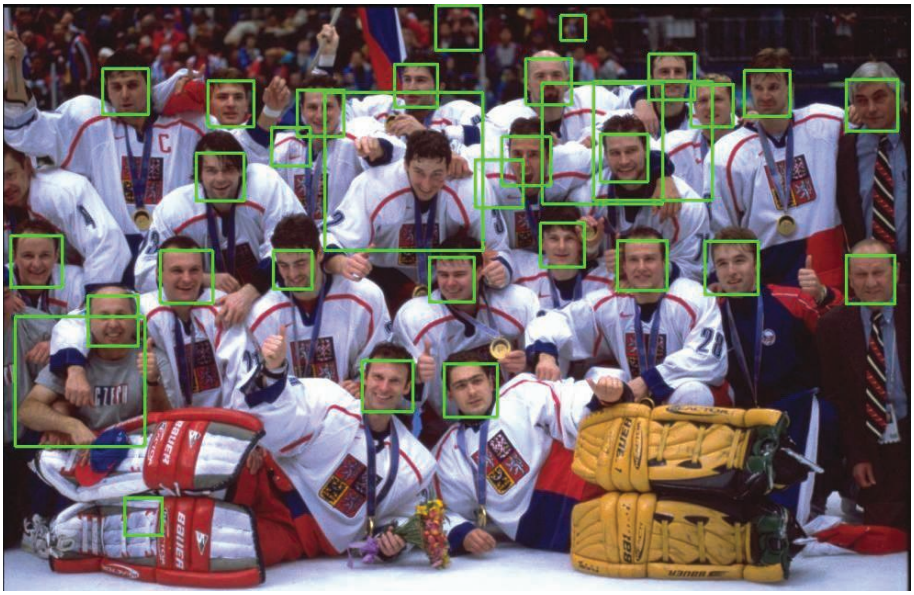
Další části mé práce je věnována Haarovým příznakům a detekcí pomocí nich. Velmi intenzivně jsem přitom využíval funkcí, které k této problematice nabízí OpenCV. Fáze učení detektoru byla vynechána, protože by byla velmi časově náročná (bylo by nutné opatřit si tisíce vzorků s tvářemi, připravit je tak, aby se na nich mohl algoritmus správně učit a poté spustit časově náročný proces učení, popsaný v kap. 5.4). Místo toho byly rovnou použity již připravené soubory z OpenCV, a to jak k detekci tváře ze předu, tak i z profilu a k detekci očí. Také mě zajímalo, jaký vliv na detekci má nastavení parametrů *min_neighbors* a *scale_factor* (více o nich v kap. 5.5.2).

Nejprve jsem vyzkoušel detekci s výchozím natrénovaným souborem *„haarcascade_frontalface_default.xml“*. Hodnota *min_neighbor* byla nastavena na 4 a *scale_factor* na 1,2. Minimální velikost tváře byla 20x20 pixelů. Detekované obličeje jsou na obr. 6.10. Vidíme, že dvě tváře zůstaly nenalezeny, naopak dvě byly detekovány vícenásobně a jednou pak bylo za tvář označeno něco úplně jiného.



Obr. 6.10: Detekce tváří výchozím souborem klasifikátorů

Abychom se pokusili algoritmus přinutit detekovat i zbylé dvě tváře a taky si vyzkoušeli vliv nastavení parametrů na hledání, změnil jsem *min_neighbors* na 2 a zmenšil *scale_factor* na 1,1 – díky tomu se bude zkoumat mnohem více podoken. Jak tato detekce dopadla je na obr. 6.11.



Obr. 6.11: Detekce výchozím souborem a upravenými parametry

Jedna z tváří stále nebyla detekována, naproti tomu počet chybně pozitivních detekcí se prudce zvýšil. Tyto parametry jsou tedy pro tento soubor nevhodné. Rozhodl jsem se pak ale vyzkoušet i další, alternativní dodávaný natrénovaný soubor *„haarcascade_frontalface_alt.xml“*. Parametry jsem ponechal stejné. Výsledek je na obrázku 6.12.

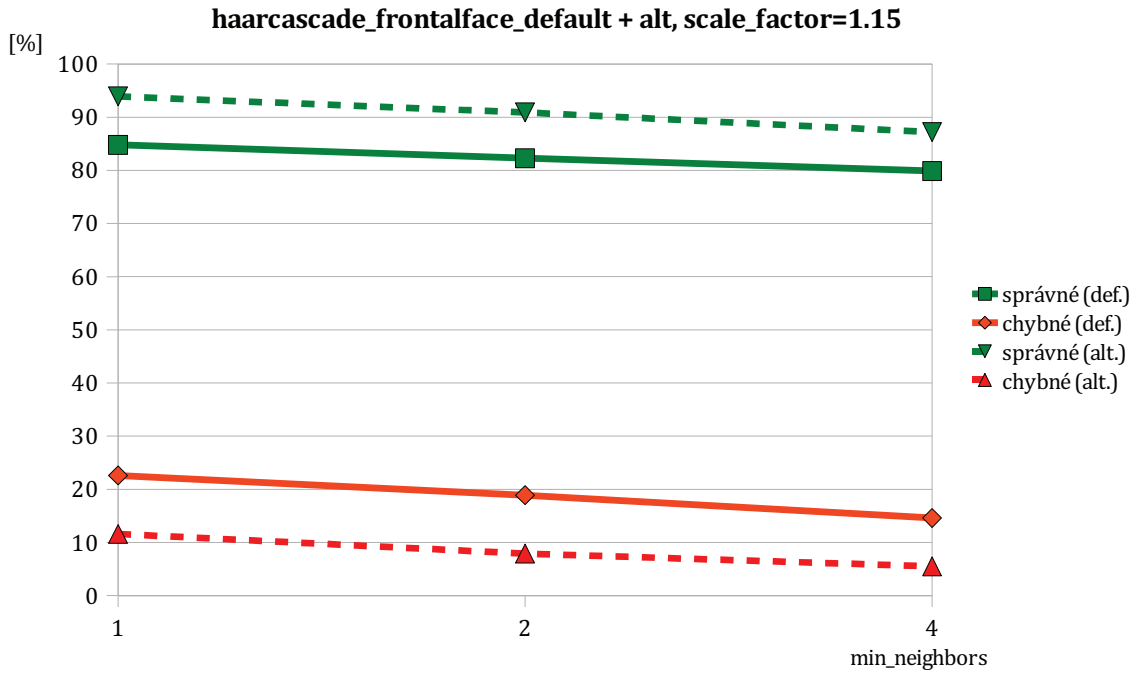
Obr. 6.12: Detekce tváří alternativním souborem

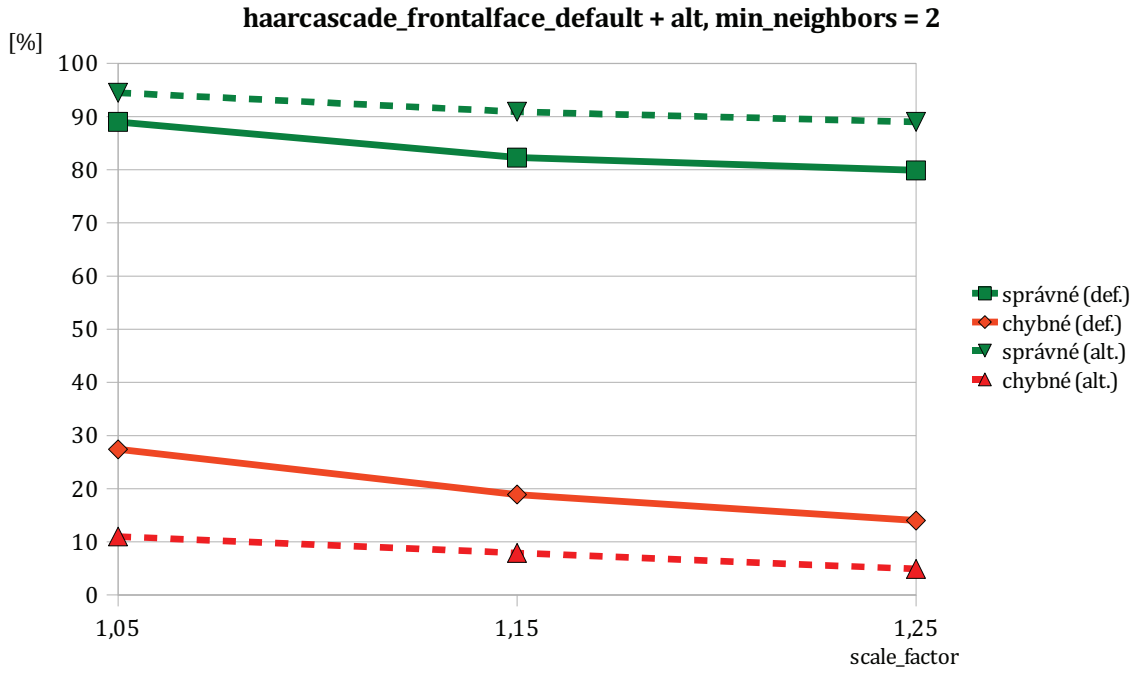
V tomto případě dopadla detekce naprosto bezchybně. Jediná tvář, která zůstala nedetekována, je úplně vlevo – není však na obrázku celá, proto se ani nelze divit, že zůstala bez povšimnutí detektoru. Z těchto výsledků vyplývá, že tento druhý (alternativní) natrénovaný soubor je kvalitnější, učení bylo asi provedeno na lepší sadě vzorků. Kromě těchto dvou souborů jsem dále vyzkoušel ještě soubor *„haarcascade_frontalface_alt2.xml“*, jeho výsledky se pohybovaly kvalitou někde mezi předchozími – jednu tvář nenašel a dvě místa chybně za tvář označil.

Jak závisí úspěšnost detekcí jsem následně ověřil sadou měření. Vytvořil jsem si testovací sadu patnácti obrázků – většinou šlo o různá skupinová foto, fotky tříd apod. Většina tváří zde byla vidět ideálně (zepředu, tváře nic nezakrývalo), ale pár bylo různě otočených, byly částečně překryty, případně osoby nosily brýle nebo měly vousy. Tyto tváře většinou činily detektoru největší problémy. Testoval jsem všechny tři výše zmíněné soubory s příznaky. Hodnotu parametru *scale_factor* jsem volil z hodnot 1,05; 1,15; 1,25. Počet *min_neighbors* byl nastavován na 1, 2 a 4. Provedl jsem tedy 27 různých měření na sadách a počítal, kolik tváří detektor správně najde a kolik míst označí chybně jako tvář. Celkový počet tváří na všech fotkách v sadě byl 164.

soubor klasifikátorů	min. neighbors	scale factor	detekovaných tváří	detekovaných tváří [%]	chybně detekovaných	chybně detek. [%]
default	1	1,05	147	89,6	53	32,3
default	1	1,15	139	84,8	37	22,6
default	1	1,25	132	80,5	26	15,9
default	2	1,05	146	89,0	45	27,4
default	2	1,15	135	82,3	31	18,9
default	2	1,25	131	79,9	23	14,0
default	4	1,05	138	84,1	32	19,5
default	4	1,15	131	79,9	24	14,6
default	4	1,25	119	72,6	18	11,0
alt	1	1,05	158	96,3	25	15,2
alt	1	1,15	154	93,9	19	11,6
alt	1	1,25	148	90,2	16	9,8
alt	2	1,05	155	94,5	18	11,0
alt	2	1,15	149	90,9	13	7,9
alt	2	1,25	146	89,0	8	4,9
alt	4	1,05	153	93,3	13	7,9
alt	4	1,15	143	87,2	9	5,5
alt	4	1,25	134	81,7	7	4,3
alt2	1	1,05	153	93,3	31	18,9
alt2	1	1,15	148	90,2	24	14,6
alt2	1	1,25	137	83,5	21	12,8
alt2	2	1,05	149	90,9	27	16,5
alt2	2	1,15	143	87,2	21	12,8
alt2	2	1,25	137	83,5	16	9,8
alt2	4	1,05	143	87,2	22	13,4
alt2	4	1,15	136	82,9	16	9,8
alt2	4	1,25	124	75,6	11	6,7

Pro lepší znázornění jsem vynesl některé výsledky do grafu. Konkrétně jde o závislosti procentuální úspěšnosti detekce pro různá *min_neighbors* při *scale_factor*=1,15 a závislost úspěšnosti pro různý *scale_factor* a konstantní *min_neighbors*=2. To vše pro výchozí soubor příznaků (v grafu plnou čarou) a alternativní (čárkovaně).



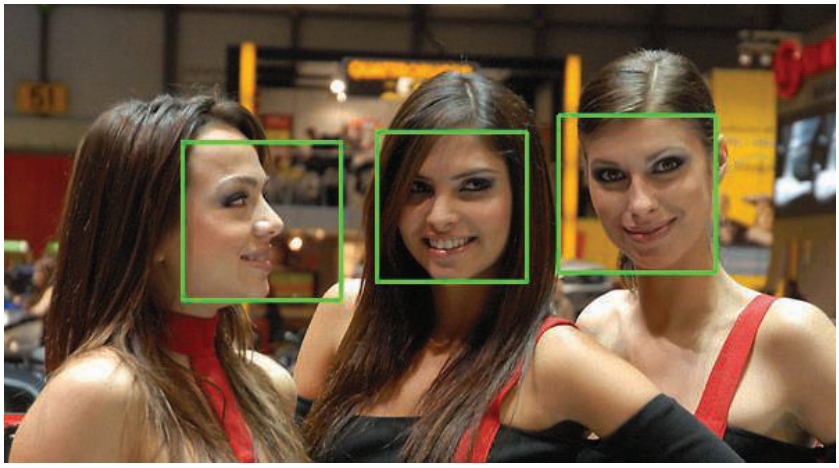


Z grafů je velmi dobře patrné, že s rostoucím *min_neighbors* klesá počet chybně označených tváří, ale i počet těch správně detekovaných. Stejného jevu si můžeme všimnout i s klesajícím *scale_factor*. Zde je rozdíl nejmarkantnější při změně z 1,05 na 1,15. Jako jednoznačně lepší soubor natrénovaných příznaků se zde ukazuje alternativní *haarcascade_frontalface_alt.xml*.

Kromě souboru pro detekci tváří zepředu obsahuje OpenCV i natrénovaný soubor pro tváře z profilu. Provedl jsem tedy jejich srovnání na obrázcích 6.13 a 6.14. Z obou obrázků je patrno, že některé tváře lze detekovat oběma soubory, některé už však ne. Některé tváře zůstaly nepovšimnuty, což však mohlo být způsobeno i nižší kvalitou fotografie. Malé tváře nebyly detekovány vůbec, jednak pro svou malou velikost a jednak prostě proto, že nebyly ani rozeznatelné. Dalším důležitým poznatkem patrným z těchto obrázků je to, že detekce založená na Haarových příznacích pracuje jen s jasovou složkou obrazu a proto funguje naprosto stejně jak na barevných, tak i na černobílých obrázcích. Následující obrázek 6.15 pak potvrzuje, že tváře snímané z profilu skutečně lze detekovat i souborem pro tváře zepředu – vysvětluji si to tím, že během trénování bylo předloženo asi i pár vzorů s tvářemi z profilu, proto je pak detektor poznal.

Obr. 6.13: Detekce tváří zepředu

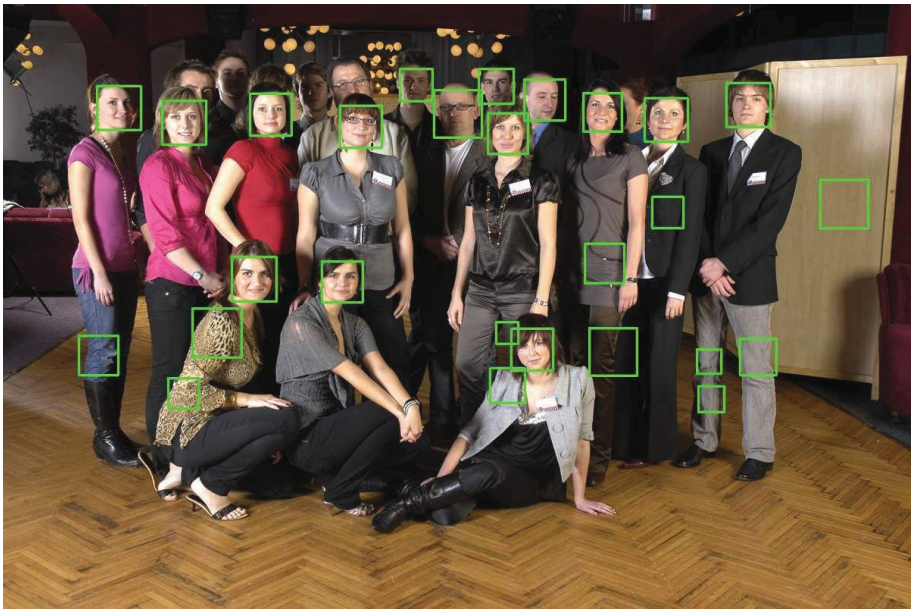
Obr. 6.14: Detekce tváří z profilu



Obr. 6.15: Detekování obou typů tváří najednou

Dále mě zajímalo, jestli by se při detekci tváří Haarovými příznaky nedalo nějak využít již hotového detektoru kůže. Hlavně tedy k odstranění chybných detekcí tváře tam, kde kůže zákonitě být nemůže. Vzal jsem tedy obrázek 3.6, kde sice detektor správně našel kůži, ale díky mnoha tvářím blízko sebe a jiným odhaleným částem těla byl jeho výsledek nepoužitelný. Na tomto obrázku jsem nejdříve pro srovnání detekoval tváře Haarovými příznaky klasicky (obr. 6.16) a posléze jsem na obrázku všechny místa nepatřící kůži zabarvil bílou barvou a detekoval Haarovými příznaky obličeje znovu (obr. 6.17). Výsledek byl velmi dobrý - všechny falešné detekce tváří se podařilo odstranit a přitom ze správně nalezených tváří neubyla ani jedna. Tím se potvrzuje využitelnost detekce kůže ne jen jako samostatné metody k detekci tváří, ale i jako

pomocné metody, která zlepšuje výsledky jiných, na jiném principu založených metod. Samozřejmě, že způsob, jakým jsem označil místa nepatřící kůži, je velmi primitivní a ne zrovna dokonalý. Mnohem lepší by bylo předkládat detektoru binární masku, podle které by se pak rozhodoval, zda vůbec na daném místě má něco zkoušet detekovat či nikoliv. Tím by se navíc samotný proces detekce i rychlil. Vyžadovalo by to ale zásahy do detektoru samotného, což by bylo asi velmi složité, proto jsem se do toho nepouštěl.

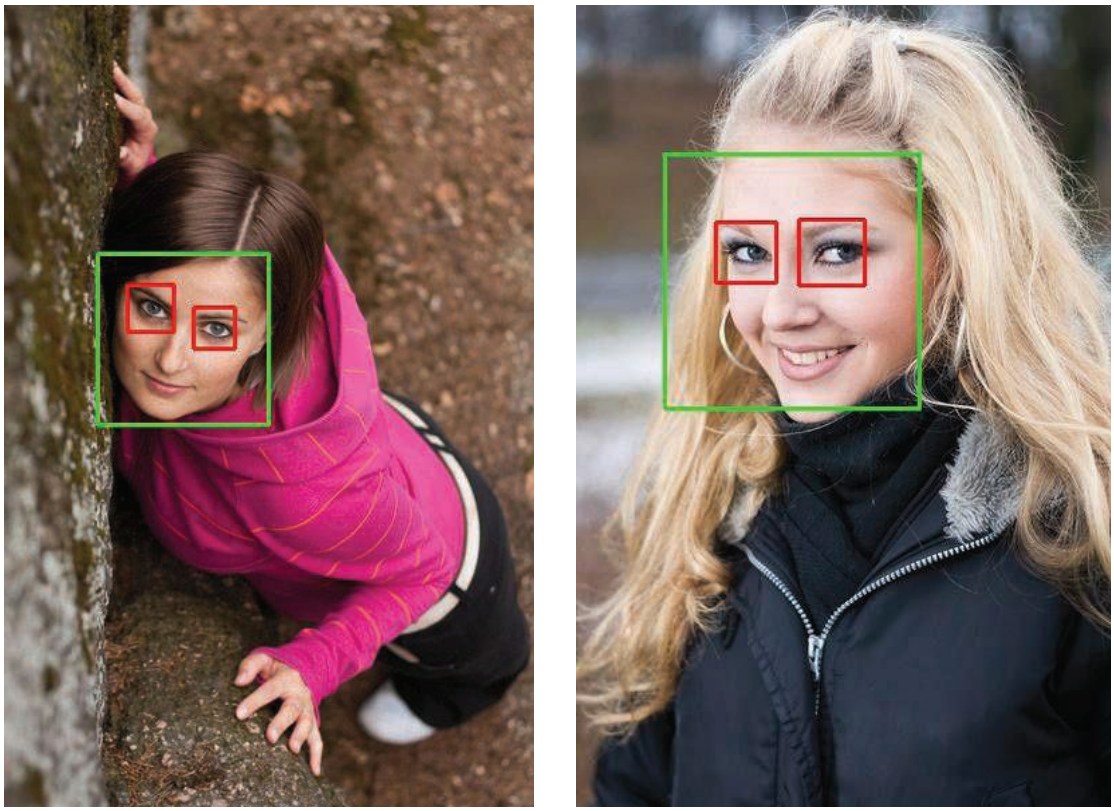


Obr. 6.16: Detekce tváří s nemaskovanými pixely

Obr. 6.17: Detekce tváří s pixely maskovanými bílou barvou

Poslední věcí, kterou jsem si vyzkoušel, byla detekce očí. I k tomu obsahuje OpenCV natrénovaný soubor (dokonce i pro detekci slunečních brýlí). Oči je možno hledat na celém obrázku, ale daleko efektivnější a lepší je nejdříve detekovat tvář a až pak na ní detekovat oči. To, že na této oblasti najdeme oči můžeme použít třeba jako ověření správnosti předchozí detekce tváře. S detekcí očí pak žádné větší problémy nebyly. Jen je třeba, aby oči byly na obrázcích rozeznatelné a např. vlivem komprese nebo rozmazání nesplývaly s okolím, případně nebyly prostě příliš malé.

Oči a ústa je možné detekovat také i barevnými metodami popsanými v kapitole 6.2. Nejprve detekujeme tvář, poté v její oblasti vytvoříme binární masku barvy kůže a nakonec detekujeme oči a ústa. Jejich umístěním a geometrií poté můžeme ověřit, zda jsme detekovali tvář, nebo je můžeme využít k detekování biometrických znaků osob.



Obr. 6.18: Detekce očí Haarovými příznaky

7 ZÁVĚR

Úkolem mé práce bylo nastudovat současné přístupy k rozpoznávání tváří a některé vybrané z nich implementovat. Protože je tato problematika velmi rozsáhlá, pokusil jsem se ji zmapovat a rozdělit existující metody do několika tříd. Těm metodám, které mi přišly zajímavé nebo perspektivní, jsem se věnoval více. Pokusil jsem se je popsat nejen teoreticky, ale uvést i příklady použití a výsledky detekce, které s nimi můžeme dosáhnout.

K implementaci jsem si nakonec vybral metody založené na detekci kůže a barevných transformací a dále učící metody založené na Viola-Jones detektoru pomocí Haarových příznaků. Důvodem výběru metod založených na detekci kůže byla jejich relativní jednoduchost a rychlost. Tyto metody s největší pravděpodobností využívají i některé fotoaparáty (i když výrobci si své tajemství chrání a žádné detaily ohledně svých technologií nezveřejňují) k detekci obličejů nebo úsměvu, protože se dají lehce implementovat na jednoduchých procesorech a jsou paměťově velmi nenáročné. U těchto metod jsem popsal barevné modely, zmínil se o jejich vhodnosti k těmto detekcím, popsal morfologické operace v obraze a i dvě asi nejpoužívanější metody k detekci očí a úst. Zmínil jsem se i o všech hlavních výhodách a nevýhodách těchto metod.

Učící metody (tzv. metody založené na zjevu) jsou již složitější a náročnější. Abychom je například mohli použít k detekci v reálném čase, potřebujeme výkonný procesor, proto je v jednoduchých zařízeních typu fotoaparát nebo mobilní telefon nemůžeme zatím použít. Jejich výhoda je však v univerzálnosti. Jednak je lze použít k detekci jakýchkoliv vzorů v obraze, nejenom tváří. Jednak také jejich síla závisí na kvalitě učení a množství klasifikátorů, které k detekci použijeme. Zvyšováním jejich počtu (a tím pádem i paměťové a časové náročnosti) zvýšíme i kvalitu detekce. Právě fázi učení jsem se snažil věnovat podrobněji, popsal jsem nejčastější klasifikátory a výpočet jejich hodnot, algoritmus učení AdaBoost, zmínil se o Haarových příznacích a o tom jak je vypočítat.

V praktické části jsem tyto metody implementoval. K tomu jsem si vybral knihovny zpracování obrazu OpenCV, které mi to velmi usnadnily. Srovnal jsem výsledky

pro různé soubory natrénovaných příznaků, vliv různých parametrů detekce na podávané výsledky a také se pokusil metody zkombinovat, abych zvýšil přesnost detekce.

Přestože problematika detekce obličejů je poměrně mladá, již nyní předvádí velmi zajímavé schpnosti, což potvrzují i výsledky, ke kterým jsem dospěl. Metody implementované v OpenCV jsou na velmi slušné úrovni a přitom stále existuje prostor k dalšímu vylepšování. Vzhledem k tomu, jak se stále rozvíjí počítačová a digitální technika, lze očekávat, že detekce objektů a tváří ještě zdaleka není na svém vrcholu a bude se v příštích letech dále intenzivně rozvíjet. Její budoucnost je rozhodně příznivá a oblastí, do kterých ještě pronikne, bude velmi mnoho a o některých dnes určitě asi nemáme ani tušení.

POUŽITÁ LITERATURA

[1] ADABOOST, Detekce objektu v obraze, Fakulta kybernetiky ZČU, Texty do kurzu přemětu Zpracování digitalizovaného obrazu. Dostupný na WWW: http://www.kky.zcu.cz/uploads/courses/zdo/lesson8/ADABOOST_cz.ppt

[2] ADABOOST, Wikipedia, the free encyclopedia. Dostupný na WWW: http://en.wikipedia.org/wiki/AdaBoost

[3] ETHEM, Alpaydin. Introduction to Machine Learning, 2004, ISBN 978-0-262-01211-9.

[4] FRIEDMAN, J., HASTIE, T., TIBSHIRANI, R., Additive logistic regression: a statistical view of boosting. Ann.Statist., Vol. 28, pp. 337--407, 2000.

[5] HLAVÁČ, V., ŠONKA M., Počítačové vidění. Grada Praha, 1992.

[6] How Face Detection Works, SERVO Magazine, February 2007. Dostupný na WWW: http://www.cognotics.com/opencv/servo_2007_series/part_2/sidebar.html

[7] JONES, M. J., REHG, J. M., Statistical colour model with application to skin detection, IEEE Int. Conf. on Computer Vision and Pattern Recognition, 1999.

[8] KIM, J. O., Kim J. S., Seo Y. R., Lee B. R., Chung C. H., Lee K. S., Yim W. Y., Lee S. H., On extraction of facial features from color images, Daegu Haany University, Kwangwoon University, Korea, 2004.

[9] LI, S. Z., JAIN A. K., Handbook of face recognition, Springer Science and Business Media, Inc., 2005, ISBN 0-387-40595-X.

[10] LIENHART, R. and MAYDT, J., An extended set of Haar-like features for rapid object detection, ICIP02, s. 900-903, 2002.

[11] LIENHART, R., LIANG, L., KURANOV, A., A Detector Tree of Boosted Classifiers for Real-time Object detection and Tracking, ICME 2003

[12] MARTINAKAUPPI, B., Face color under varying illumination – analysis and applications. Dissertation work, University of Oulu, Oulu university press 2002.

[13] OpenCV Documentation, OpenCV Experimental Functionality. Dostupný na WWW: http://www710.univ-lyon1.fr/~bouakaz/OpenCV-0.9.5/docs/ref/OpenCVRef_Experimental.htm

[14] OSUNA, E., FREUND, R., GIROSI, F. Training Support Vector Machines: An Application to Face Detection. In Computer Vision and Pattern Recognition, 1997..

[15] PŘINOSIL, J., Krolikowski M., Využití detektoru Viola-Jones pro lokalizaci obličeje a očí v barevných obrazech, Elektrorevue ISSN 1213 – 1539, VUT Brno, 2008.

[16] ŘÍHA, K., Pokročilé techniky zpracování obrazu. Skripta k předmětu MPZO, VUT Brno, 2007.

[17] VIOLA, P., JONES, M. Robust Real-time Object Detection. July 13, 2001. Vancouver, Canada. Dostupný na WWW: http://cs223b.stanford.edu/notes/viola01robust.pdf

[18] VLACH, J., Přinosil J., Lokalizace obličeje v obraze s komplexním pozadím. Elektrorevue ISSN 1213 – 1539, VUT, Brno 2007.

[19] YANG, G., HUANG, T. S. Human Face Detection in Complex Background, In Pattern Recognition, Volume 27., 1994.

[20] YANG, M. H., Ahuja N., Detection human faces in color image. In Proc. of IEEE International Conference on Image Processing, volume 1, 1998.

[21] YOUNGMIN, L., Gong S., Sherrah J., Multi-view face detection using SVM and Eigenspace modelling, University of London. Dostupný na WWW: http://reference.kfupm.edu.sa/content/m/u/multi_view_face_detection_using_support__241769.pdf

[22] van de WETERING, H. M. M., Eye Tracking, January 21, 2008. Dostupný z WWW: http://www.hanckmann.net/userfiles/Gaze%20Tracking%20Report.pdf

PŘÍLOHY

Algoritmus výpočtu EyeMapC:

Definice funkce:

```
020 IplImage* EyeMapC(IplImage* ImageZdrojovy, IplImage* ImageMaska) {
```

Příprava výstupu, konverze do YCbCr prostoru:

```
022     CvSize velikost = cvSize(ImageZdrojovy->width, ImageZdrojovy->height);
023     IplImage* vysledek = cvCreateImage(velikost, 8, 1);
024     IplImage* ycbcr = cvCreateImage(velikost, 8, 3);
025     cvCvtColor(ImageZdrojovy, ycbcr, CV_RGB2YCrCb);
```

Normalizace jednotlivých složek na plný rozsah hodnot:

```
028     _normalizace(ycbcr, ImageMaska, velikost.width, velikost.height, 3, 0);
029     _normalizace(ycbcr, ImageMaska, velikost.width, velikost.height, 3, 1);
030     _normalizace(ycbcr, ImageMaska, velikost.width, velikost.height, 3, 2);
```

Cyklus procházející jednotlivé pixely, který počítá výslednou mapu:

```
032     for (int i=0; i<velikost.width; i++) for (int j=0; j<velikost.height; j++)
          if (CV_IMAGE_ELEM(ImageMaska, uchar, j, i) != 0) {
033         int cb = CV_IMAGE_ELEM(ycbcr, uchar, j, i*3+1);
034         int cr = CV_IMAGE_ELEM(ycbcr, uchar, j, i*3+2);
035         CV_IMAGE_ELEM(vysledek, uchar, j, i) =
                (cb*cb + (255-cr)*(255-cr) + cb/(cr+1))/1000;
036     } else CV_IMAGE_ELEM(vysledek, uchar, j, i) = 0xff;
```

Uvolnění konvertovaného obrázku, normalizace výsledku a ukončení:

```
038     cvReleaseImage(&ycbcr);
039     _normalizace(vysledek, ImageMaska, velikost.width, velikost.height, 1, 0);
040     return vysledek;
041 }
```

Výpočet EyeMapL:

Definice funkce:

```
043 IplImage* EyeMapL(IplImage* ImageZdrojovy, IplImage* ImageMaska) {
```

Vytvoření pomocných obrázků a konverze do YCbCr barev:

```
045     CvSize velikost = cvSize(ImageZdrojovy->width, ImageZdrojovy->height);
046     IplImage* vysledek = cvCreateImage(velikost, 8, 1);
047     IplImage* ycbcr = cvCreateImage(velikost, 8, 3);
048     IplImage* dil = cvCreateImage(velikost, 8, 1);
049     IplImage* ero = cvCreateImage(velikost, 8, 1);
050     cvCvtColor(ImageZdrojovy, ycbcr, CV_RGB2YCrCb);
```

Je potřeba normalizovat jasovou složku:

```
053     _normalizace(ycbcr, ImageMaska, velikost.width, velikost.height, 3, 0);
```

Překopírování jasové složky do pomocných obrázků:

```
055     for (int i=0; i<velikost.width; i++) for (int j=0; j<velikost.height; j++) if
          (CV_IMAGE_ELEM(ImageMaska, uchar, j, i) != 0) {
```

```
056         CV_IMAGE_ELEM(dil, uchar, j, i) = CV_IMAGE_ELEM(ycbcr, uchar, j, i*3);
057         CV_IMAGE_ELEM(ero, uchar, j, i) = CV_IMAGE_ELEM(ycbcr, uchar, j, i*3);
058     } else {
059         CV_IMAGE_ELEM(dil, uchar, j, i) = 0xff;
060         CV_IMAGE_ELEM(ero, uchar, j, i) = 0xff;
061     }
```

Základem této funkce jsou morfologické operace s kruhovým strukturním elementem (kvůli tvaru zorniček), protoje třeba jej vytvořit:

```
064   IplConvKernel* jadro = cvCreateStructuringElementEx(5, 5, 2, 2, CV_SHAPE_ELLIPSE, 0);
```

Provedeme dilataci a erozi podle rov. 3.11:

```
067     cvDilate(dil, dil, jadro, 1);
068     cvErode(ero, ero, jadro, 1);
071     for (int i=0; i<velikost.width; i++) for (int j=0; j<velikost.height; j++) if
          (CV_IMAGE_ELEM(ImageMaska, uchar, j, i) != 0) {
072         CV_IMAGE_ELEM(vysledek, uchar, j, i) = CV_IMAGE_ELEM(dil, uchar, j, i)/
              (CV_IMAGE_ELEM(ero, uchar, j, i)+1);
073     } else CV_IMAGE_ELEM(vysledek, uchar, j, i) = 0xff;
```

Normalizace výsledku, uvolnění dočasných obrazů a konec

```
081     _normalizace(vysledek, ImageMaska, velikost.width, velikost.height, 1, 0);
083     cvReleaseStructuringElement(&jadro);
084     cvReleaseImage(&ycbcr);
085     cvReleaseImage(&dil);
086     cvReleaseImage(&ero);
088     return vysledek;
089 }
```

Výpočet MouthMap:

Definice:

```
091 IplImage* MouthMap(IplImage* ImageZdrojovy, IplImage* ImageMaska) {
```

Vytvoření pomocných obrázků a převod do YCbCr prostoru:

```
092     CvSize velikost = cvSize(ImageZdrojovy->width, ImageZdrojovy->height);
093     IplImage* vysledek = cvCreateImage(velikost, 8, 1);
094     IplImage* vysledek32 = cvCreateImage(velikost, 32, 1);
095     IplImage* ycbcr = cvCreateImage(velikost, 8, 3);
096     cvCvtColor(ImageZdrojovy, ycbcr, CV_RGB2YCrCb);
```

Normalizace Cb a Cr složek:

```
099     _normalizace(ycbcr, ImageMaska, velikost.width, velikost.height, 3, 1);
100     _normalizace(ycbcr, ImageMaska, velikost.width, velikost.height, 3, 2);
```

Výpočet sum dle rovnice, abychom mohli určit η:

```
102     int n = 0;
103     int suma_cr = 0;
104     int suma_crcb = 0;
105     int cr, cb;
106     for (int i=0; i<velikost.width; i++) for (int j=0; j<velikost.height; j++)
          if (CV_IMAGE_ELEM(ImageMaska, uchar, j, i) != 0) {
107         n++;
108         cb = CV_IMAGE_ELEM(ycbcr, uchar, j, i*3+1);
109         cr = CV_IMAGE_ELEM(ycbcr, uchar, j, i*3+2);
110         suma_cr += (cr*cr)/255;
```

```
111         suma_crcb += cr/(cb+1);
112     }
```

Výpočet η a finálního obrázku (s 32bit. přesností):

```
114     float eta = (float)0.95*suma_cr/suma_crcb;
115     for (int i=0; i<velikost.width; i++) for (int j=0; j<velikost.height; j++)
          if (CV_IMAGE_ELEM(ImageMaska, uchar, j, i) != 0) {
116         cb = CV_IMAGE_ELEM(ycbcr, uchar, j, i*3+1);
117         cr = CV_IMAGE_ELEM(ycbcr, uchar, j, i*3+2);
118         CV_IMAGE_ELEM(vysledek32, float, j, i) = (float)((cr*cr)*(cr*cr-eta*cr/
              (cb+1))*(cr*cr-eta*cr/(cb+1)));
119     } else CV_IMAGE_ELEM(vysledek32, float, j, i) = 0.0;
```

Normalizace výsledku a jeho převod zpět na 8bit obrázek:

```
121     cvNormalize(vysledek32, vysledek32, 0.0, 1.0, CV_MINMAX, NULL);
122     for (int i=0; i<velikost.width; i++) for (int j=0; j<velikost.height; j++)
          if (CV_IMAGE_ELEM(ImageMaska, uchar, j, i) != 0) {
123         CV_IMAGE_ELEM(vysledek, uchar, j, i) =
              cvFloor(255*CV_IMAGE_ELEM(vysledek32, float, j, i));
125     } else CV_IMAGE_ELEM(vysledek, uchar, j, i) = 0xff;
```

Uvolnění dočasných obrázků a vrácení výsledku:

```
127     cvReleaseImage(&ycbcr);
128     cvReleaseImage(&vysledek32);
129     return vysledek;
130 }
```

Detekce obličejů Haarovým detektorem

Alokace paměti pro detektor:

```
123     CvMemStorage* VysledkyHledani = cvCreateMemStorage(0);
124     cvClearMemStorage(VysledkyHledani);
```

Načtení natrénovaného souboru s klasifikátory:

```
127     CvHaarClassifierCascade* HaarPriznaky =
          (CvHaarClassifierCascade*)cvLoad("haarcascade_frontalface_alt.xml", 0, 0, 0);
```

Vytvoření proměných pro výsledky a nastavení minimálního prohledávaného podokna:

```
129     CvSeq* Obliceje;
130     CvSize MinVelikost = cvSize(20, 20);    //minimalni velikost obliceje
```

Spuštění hledání:

```
133     if (Obliceje = cvHaarDetectObjects(ImageObliceje, HaarPriznaky, VysledkyHledani,
          1.1, 2, CV_HAAR_DO_CANNY_PRUNING, MinVelikost)) {
```

Procházení jednotlivých nalezených oblastí a vykreslení jejich ohraničení:

```
136         for (int i=0; i<Obliceje->total; i++) {
137             CvRect* r = (CvRect *)cvGetSeqElem(Obliceje, i);
138             cvRectangle(ImageObliceje, cvPoint(r->x, r->y), cvPoint(r->x+r->width,
                  r->y+r->height), CV_RGB(0, 255, 0), 2, 8, 0);
139         }
140     }
```